Mastering the Data Pipeline for Autonomous Driving

Patrik Moravek[1] and Bassam Abdelghani1[1]

[1]dSPACE GmbH, Rathenaustraße 26, 33102, Paderborn, Germany
Pmoravek@dspace.de, babdelghani@dspace.de

Abstract.
Autonomous driving is at hand, for some at least. Others are still struggling to produce basic ADAS functions efficiently. What is the difference between the two? It is how the data is treated and used. The companies on the front line realized long ago that data plays a key and central role in the progress and development processes must be adapted accordingly. Those companies that have not adapted their processes are still struggling to catch up and are wasting time and resources.

This article discusses the key aspects and stages of data-driven development and points out the most common bottlenecks. It does not make sense to focus on just one part of the data-driven development pipeline and neglect the others. Only harmonized improvements along the entire pipeline will allow for faster progress. Inconsistencies in formats and interfaces are the most common source of project delays. Therefore, we provide a perspective from the start of the data pipeline to the application of the selected data in the training and validation processes and on to the new start of the cycle. We address all parts of the data pipeline including data logging, ingestion, management, analysis, augmentation, training, and validation using open-loop methods.

The integrated pipeline for the continuous development of machine-learning-based functions without inefficiencies is the final goal, and the technologies presented here describe how to achieve it.

Keywords: data analysis; data augmentation; data collection; data management; data pipeline; data replay; data recording; data retrieval; data selection; ingestion pipeline; reprocessing; training; validation.

1 Introduction

Big data is not what is standing in the way of the automotive industry's attempt to automize driving. It is the way the data is used and harnessed that is hindering progress toward driverless mobility. It is easy to collect a lot of data, it is easy to store this data, and it is easy to retrieve the data. However, it is not easy to do this efficiently. Efficient means not wasting resources and time. Data is great and it is considered "the new oil," but too much oil can choke the engine and even cause the engine to

stall. Therefore, it is not the data itself but the process and appropriate tools that advance the progress.

There is a huge amount of data being collected by the vehicle sensors, adding up to dozens of terabytes per vehicle per day. And while the total data lake can reach hundreds of petabytes (e.g., BMW´s D3 data center features more than 200 PB of storage [1]) current estimations say only 5% of the data is actually accessed and used productively.

Reliable autonomous operation is an ambitious goal due to the complexity of the problem to solve. It is not a well-defined problem within a limited scope that would allow traditional problem-solving approaches to succeed. The complexity of the environment, the real world, in which autonomous vehicles must be operated productively is enormous. And even an artificial limitation of the relevant conditions such as a restricted area or weather conditions (often defined in ODD – Operative Domain Description) does not make it possible to describe explicitly intended operation in the form of if-then-else. That is the reason for which a completely different approach to solving this problem has been adopted. It is a general heuristic approach that does not intend to solve the problem with the highest accuracy and completeness, but rather aims to solve the problem pragmatically in an efficient manner. It has been proven that a more general code framework that can be configured and tuned with specific examples of the problem to solve results in significant advances in autonomous driving. It can be called programming by examples or, as Andrej Karpathy, the director of AI development at Tesla, calls it, SW 2.0 [2].

The clear separation in naming (SW 2.0 vs. classic software) is related to the completely new paradigm in programming related to the problem-solving method. It is no longer the written code that defines whether the program successfully solves the task. Rather, the parametrization of the code has the most significant impact.

In conventional software development, a programmer writes instructions line by line to define what is executed automatically. In the new approach, only a few lines of codes are written that define a model and parameters of the model that are set afterward. The success or failure of the program thus depends to a very large extent on the configuration of the parameters. The parameters are not set manually. There are thousands or even millions of parameters that have to be adjusted. Fortunately, this is done automatically by describing the right examples of how the output should relate to an input. In other words, the model is fed input data and its output is compared with the expected output. The difference is back propagated in the model, adapting the parameters to achieve a smaller difference in the next run. After this optimization task and many different iterations, the model parameters are set to deliver an output that is "sufficiently close" to the expected result. The model is then adjusted to the provided examples (data) and returns the expected results. This is good because the model automatically returns the expected output based on the provided input without having to explicitly define every step in the process. Only the model is hard-coded, while the parameters are defined automatically. The issue is that the model will work as long as the provided input corresponds to the examples given to the model during the parametrization (training of the model). If the model has not a seen similar input, it will

most likely not deliver the expected output. The consequence is that the success of the development is determined by the data the model sees during the training process.

Due to this fact, the roles in the development process are split between programmers who write the code to implement the model and data engineers who prepare the data for the training phase. After the initial phase, the progress of the project is meas-ured based on how much the training data, and to a lesser extent the code, has im-proved. This means that the focus of the development optimization changes complete-ly. The focus is not on code advances, code versioning, or integration with all the tools that help streamline the code development and testing process. Completely new requirements for supporting tools have emerged. The development process must adopt this fundamental change of focus and place data at the center of attention. The pro-cesses must be wrapped around data and allow for the efficient use of the data in vari-ous development steps. The environment must be able to handle large amounts of data as the trained model must not be surprised by new situations (input) on the road.

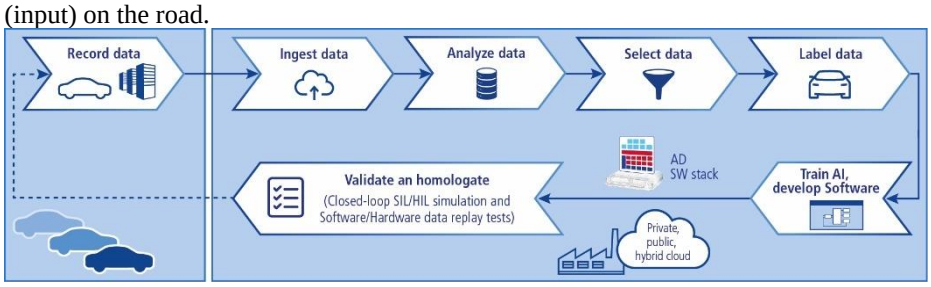

Fig. 1. Data pipeline and the cycle of data-driven development in the automotive industry

The structure of the paper is aligned with the main domains of the data pipeline and the data cycle supported by dSPACE (as depicted in Fig. 1). Each chapter below is dedicated to one key topic. Nevertheless, since the main narrative of the data pipeline is "seamlessness," the interconnections and mutual influences are highlighted in the individual chapters.

2 Onboard Data Pipeline

The journey of the data through the data pipeline starts at the vehicle. The vehicle is the first hub and the source of the data to be used to advance autonomous driving functions. One vehicle can be seen as a single source of data when looking at it from the perspective of a data collection campaign including several vehicles. At the same time, the vehicle itself represents a data collection network. At the vehicle level, the data sources are all sensors, buses, and networks that transmit information from internal vehicle sensors and ECUs. Simple recording simply means shuffling the data from IO interfaces to data storage as fast as possible. More advanced logging introduces functions that are applied to the data between their reception and their storage. This is

a data pipeline at the vehicle level. Both perspectives have their own challenges and aspects, and thus they can be split up for further consideration.

Acquisition of data at the vehicle level has become yet another challenging topic in the general pursuit of autonomy. Unlike legacy acquisition technologies, data logging systems for ADAS/AD must cope with new interfaces, protocols, high data rates, and potentially huge data volumes produced by environmental sensors.

The vehicle and the data collection system are also the first place where the data pipeline can be optimized. The internal vehicle bus and network architecture can be quite complex, and failures or inconsistencies can occur when acquiring data. This is not infrequent and such failures must be identified early on in order to avoid wasting time (when collecting corrupted data) and storage space.

The most common practice currently is for in-vehicle data logging systems to collect all of the data and for the quality of the data to be checked while ingesting the data or in the data center. This method wastes the resources in the vehicle. Most obviously, the size of the in-vehicle storage must compensate for the inefficiency. In addition, the drives must be interrupted more frequently than necessary just to swap drives or upload the data from the vehicle to an ingestion station.

This inefficient process has been identified, and the emerging trend is to equip vehicles with an additional computer to analyze the incoming data while driving. This is an intermediate step as it is not optimal due to the increased complexity of the vehicle measurement equipment. An additional computer requires additional power, space, mounting, a high bandwidth connection, and precise synchronization with the rest of the system. The final goal is for the data logger itself to be able to analyze the incoming traffic and decide what is to be recorded and what is to be thrown away. This poses additional requirements for the data logger to supply enough computational power for the analysis. In addition, the logger has to offer efficient means to set up such a data analysis in the software framework or logging application itself.

Several levels of detail can be distinguished when curating and analyzing data. First, the most basic requirement is to detect any missing data streams, which would indicate a problem with an in-vehicle component. This might happen if some of the sensors or ECUs do not boot properly after restarting the system. Additionally, the data rate of the data streams can be checked to indicate configuration issues. Analyzing the headers and content of the received packages and messages offers far-reaching possibilities for understanding the conditions and surroundings of the ego vehicle.

Such a content-based analysis not only checks the data quality but also optimizes the entire data cycle. If the data logger is situation-aware, relatively simple rules can be defined to improve the data flow. Such rules can trigger the tagging of the data for later use, define filters on the data logging pipeline, or enable a triggered recording to store only data of a defined interest.

The triggered recording can be enhanced with an over-the-air transmission feature. In this case, the data is not only stored in the vehicle storage but immediately sent out to a data center to be available for data scientists as soon as possible. Such a data flow shortens the entire data cycle as much as possible and allows for the accelerated improvement of the neural networks employed.

Achieving in-depth insight into the data from the data logger has several prerequisites. First, the data logging has to be centralized to have a comprehensive view of the data from all devices. Then, it is necessary to provide support for all interfaces (CAN, FlexRay, Ethernet, GMSL, etc.) and understand the communication protocols to access the signal and data level. And potentially, it must be possible to interpret the signals and data themselves.

Interpreting signals is quite trivial (assuming it is possible to read the communication matrix), while interpreting data from complex environmental sensors is the challenging part. It is exactly what ADAS/AD ECUs are intended for with the difference that the reliability requirement is much lower. Much lower levels of perception are accepted. The detection is only for informative purposes and does not have any safety implications. False positives might just mean a bit more data is recorded. False negatives at further distances are not as much of a worry as buffering is in place anyway. Once the objects are closer, the probability of detection increases, and a specified interval before the detection is recorded. It is not necessary to understand the complete situation. Only certain aspects of the situation, certain objects, or certain maneuvers are of interest in the particular phase of development and testing. This simplifies the logging configuration and the in-vehicle data pipeline. In addition, it can be dynamically adapted based on the actual needs.

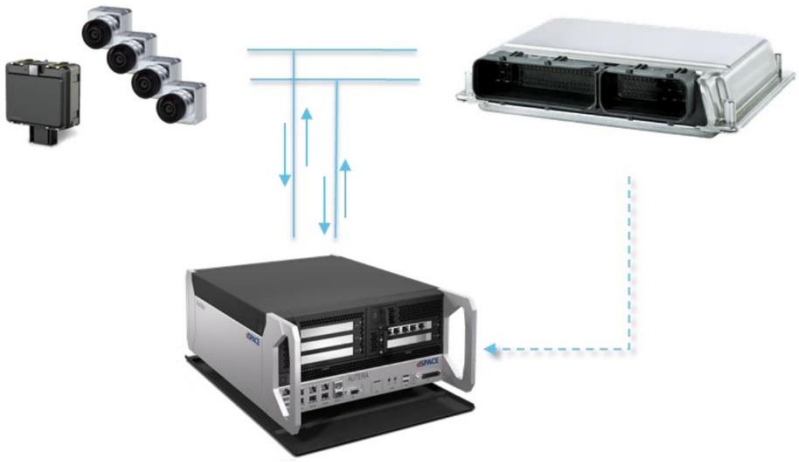

Fig. 2. Logging data during road testing involves a specific logging method that incorporates a minimal level of delay and jitter to not affect the system under test

The road testing phase provides another option to optimize the data logging process. The combination of a performant computing platform and a data logger in one device makes it easy to deploy a perception algorithm together with the logging pipeline. Certain KPIs for the algorithm under test (e.g., uncertainty level) can act as a

trigger to store the relevant time interval in the recording and even to send it over-the-air to the data center to speed up the feedback loop.

When an ADAS/AD computer (or its prototype) is under test in the vehicle, the computational and logging platforms are separated but the same synergy effect can be achieved (see the data logging architecture in **Fig. 2**). KPIs for algorithms under test can be a trigger to the recording. In this case, not only the sensor and bus data are saved but also the state of the ECU under test can be saved consistently. This allows for detailed and efficient debugging with a complete data snapshot of the situation. Furthermore, it is possible not only to log externally exposed function parameters but also to store the memory dump.

3 Ingestion Pipeline

Ever improving environmental sensors with increasing resolution produce ever increasing amount of data. Cameras in particular are the drivers for data bandwidth and storage requirements. Parking cameras might feature 1.2 or 2 Mpx of resolution, but the imager in more common cameras found in SEA L2+ vehicles provide 5 to 8 MPx of resolution. Recently, the company Ambarella announced that it would be launching an Intelligent AI Vision processor capable of processing outputs of 8k resolution cameras at up to 60 frames per second with low consumption [3]. This will make it possible to introduce cameras with an 8k resolution in the automotive domain as well (unlike the 4k cameras seen today). If data from such cameras were to be logged in a raw form (as it is currently done), a single 8k camera running at 30 frames per second would require a recording bandwidth of at least 11.5 gbps. Currently, typical data rates (aggregated) range from a few gigabits per second to 40-50 gbps, making it possible to file even large storage drives very quickly.

When in-vehicle storage (usually SSD or HDD types) is full, data must be transferred to a storage location that can be accessed by developers and test engineers. If we ignore the over-the-air transmission of selected sequences for now, there are two basic means of transferring collected data. One option is to ingest data from in-vehicle storage to a data center directly. Directly means that the drives (SSDs or HDDs) are transported to the data center where the data is finally stored. The data is copied from the in-vehicle storage to the server storage, while the drives are erased and returned to the drivers. The second option is to not transport the storage drives themselves but rather to transmit the recorded data. If all available storage drives in the vehicle are full, the driver drives to a place with a good connection, and the data is transmitted via the Internet (usually via dedicated links with high bandwidth such as Express Route Direct from Microsoft [4]). Both approaches have their pros and cons, and the goal is to optimize the cost/transfer time ratio.

Whatever approach is selected, the ingestion pipeline starts by reading the data from the vehicle drives. As a rule, data that cannot (or will not) be used should not be ingested. This is especially true for corrupted or incomplete data. Such data would only take up space without any added benefit. If the quality check is not performed in

the vehicle, the ingestion pipeline is another possibility for maximizing storage and thus saving costs.

In addition to the bandwidth requirements, implementing an ingestion pipeline can be computationally demanding. Therefore, the processing takes place in the data center itself or in what is referred to as a collocation center. Such facilities offer much cheaper computational power than is available in the vehicle and have fewer restrictions in terms of space and power supply.

The ingestion pipeline is an abstraction of the hardware and location where it is run. It represents logical steps that the data goes through to become available for users.

Quality control is the first stage in the ingestion pipeline. It is a step that pays off as low-quality data does not have to be ingested because it will never be used and thus can be deleted. Low-quality refers to incomplete data, corrupted data, inconsistent data, or data with an error flag (from the data source).

Automated annotations are another stage. The primary purpose of annotation in this step is to provide a possibility to analyze and search for data early on. This allows for more informed decision-making concerning further data processing. At this stage, annotation functions are executed to derive basic information about the data. Usually, the functions are not power hungry as they process all data (instead of only preselected data). There might be different strategies, but easily derived annotations conducted in this phase include map-based annotations, weather and day-time annotations, and annotations derived directly from bus data.

After this stage, users can analyze statistical information based on the data: for example, the distribution of road types, road infrastructure, or weather conditions such as rain, fog, and storms. In addition, information about traffic signs is extracted from the map data to provide further insight on road sections of interest.

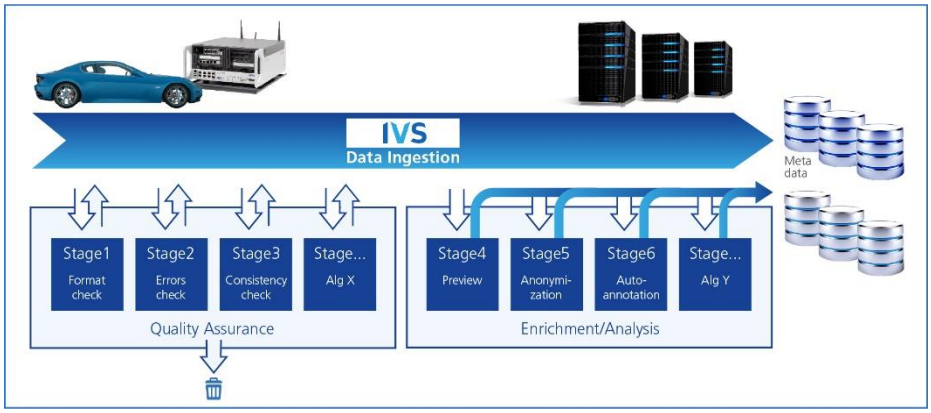

Fig. 3. Data ingestion pipeline ensuring only quality data are ingested and automatically preparing data for searches and analyses in compliance with GDPR

Another step involves the preparation of data for visualization. Users expect detailed visualization options that will allow them to see the vehicle route on a map and to

preview data from different sensors. Therefore, highly compressed previews optimized for web visualization are generated for every recording. In addition, the previews can be anonymized (faces and license plates are blurred or artificially generated) to comply with GDPR regulations and to simplify the organizational measures.

Ingestion can be executed fully automatically as a part of the overall data pipeline. It all starts with the preconfigured upload station. At the push of a button, the data is retrieved from the vehicle SSD (or hard drive) and processed by the ingestion pipeline functions to prepare it without any human interaction. At the end of the pipeline, the test or data engineers are able to access, preview, and search in the data to accomplish their tasks.

4 Data Annotation, Analysis, and Selection

In a broader sense, annotation is the extraction of structured information from unstructured data. It is necessary to make the unstructured data easier to understand and use. In the context of autonomous driving, raw video, lidar, and radar data is considered unstructured. Annotations are metadata and, to a certain extent, they result from a processing stage that makes it possible to execute higher level abstraction algorithms on the data itself. For example, when calculating the number of objects in a scenario or calculating the distribution of distances to other objects, it is first necessary to recognize the objects. Another example is searching for a particular aspect or situation. Running a search on the raw data is not feasible. Therefore, metadata extracted from the raw data is used to improve search responsiveness.

An overview of the purposes and requirements of annotations in the process of data-driven development is shown below.

- **Filtering and reducing data**

  Reducing the volume of data to data that significantly adds value is one of the main goals of data-driven development. Annotations make it possible to define rules for in-vehicle or ingestion pipelines that either delete the recorded data or speed up the utilization of data by sending it off the vehicle via wireless links.

  Any annotation type can be used for filter rules and their quality requirements may differ. If a filter is used to preselect data, some false positive bias is allowed as it “only” increases the used wireless bandwidth. If a filter is used to reduce data, more confidence is required in the annotation to not delete useful situations that do not have sufficient representation in the already collected dataset.

  One efficient way of reducing the data volume for AI training is to eliminate frames that do not balance the training data set; frames that are too similar to others. A specific description vector of a frame is used in this case as an annotation and compared with vectors from other frames. For more details, see [5].

- **Analyzing data**

  Without annotation, a great deal of important metadata would be missing. By annotating the data, it is possible to gain insight into the distribution of different

aspects of interest such as highway versus rural roads, the number of merge points on the highway, or the number of surrounding vehicles (see example graphic in **Fig. 4**).

Annotations for data analysis are also necessary to evaluate whether set goals are achieved in data logging campaigns. A predefined number of recordings for a specific situation is usually the goal that determines the length of the campaign.

- **Searching for and selecting data**

  Possessing data without being able to find something efficiently is a waste of resources. Annotations can be used to quickly search without intensive computational tasks being performed every time data is requested from a data lake.

  Building on the search, data can be structured and organized to prepare dedicated purpose-built datasets to facilitate the execution of further stages in the data pipeline like labeling, augmentation, and even AI training and validation processes.

  The most common search query is a text string describing a particular aspect of interest (e.g., night, rain, bicycle, distance <50m, etc.). During AI validation, usually some particular situation or frame causes the trained model to fail and the remedy is to expand the training dataset. Therefore, the search aims to find data that is similar to the problematic frame. Instead of describing the picture in human terms (e.g., urban situation, rain, etc.) the problematic frame itself can be used as a query that looks for similar pictures or situations. This innovative and promising approach is described in more detail in [6].

- **AI training**

  AI training requires high-quality annotations (also called labels). The more precise the labels, the better the training result and the better the performance of the final ADAS/AD function [7]. For perception functions, both static and dynamic objects need to be annotated with their size, position, orientation, and other aspects to provide sufficient ground-truth information.

- **Validation**

  Validation always needs a reference. When validating perception and fusion functions, annotations serve as this reference (i.e., ground truth). As the name implies, the ground truth is a precise description of reality that is captured in the sensor data.

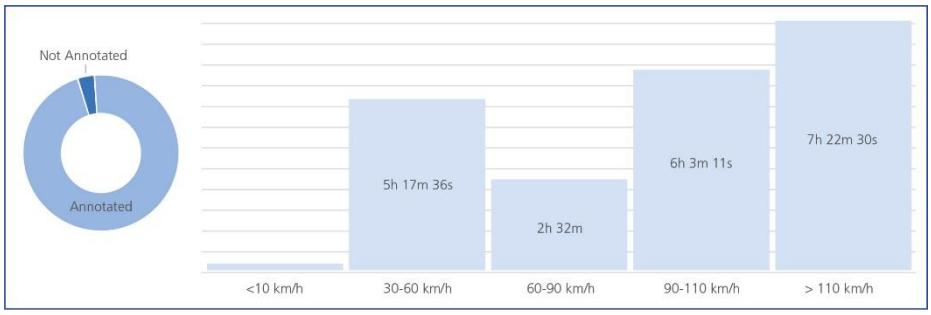

Fig. 4. Example of distribution analysis for data speed road sections

There are several types of annotations that are extracted and used in the process of data-driven development. Often the term "tag" is used for situational annotations describing general aspects of the situation such as rainy or pedestrian crossing and the term "label" is used to identify the object in the data precisely. However, there are overlaps and such differentiated terminology is not always unequivocal.

- **Manual annotations** – driver or assistant manually adds whatever kind of information that can be used for the purposes mentioned above (usually not the annotations for reference data).
- **Ego vehicle tags** – these tags are derived from ego parameters extracted during the analysis of bus and network communication (e.g., speed, acceleration, yaw angle)
- **Weather and day-time annotations** – information about the actual weather situation, for example, from public services that provide the information based on the position and time or by analyzing camera data.
- **Map-based annotations** – maps are a rich source of useful information that can serve the purpose of annotation. Road types, road infrastructure, traffic signs, and speed limits are just a few examples of possible extracted tags.
- **Feature vector** – this is a special type of annotation calculated based on the picture and used for the similarity search.
- **Object detections and object properties** – identification of objects in the scene and their properties based on neural networks or derived from manual data. Examples of objects include static objects such as guardrails and streetlamps and dynamic objects such as other traffic participants (e.g., pedestrians, trucks, buses, etc.)

It is also worth mentioning that annotations are not free and entail certain costs. If the annotations are extracted automatically, then the cost of computational resources is considered. If a human annotator performs the job, then the personnel costs are considered. For high-quality annotations, used for AI training and validation, the combination of automated process and human quality controls is often the preferred method as it optimizes the costs.

5 Data Replay

Autonomous vehicles can only gain the trust of consumers if the technology is proven to work in all cases, emphasizing the role of automated testing. Key technologies for autonomous driving such as environment perception and sensor fusion require a different approach to testing. Data replay, a relatively new testing strategy, has established itself as an efficient and cost-effective testing and validation method for these technologies.

Various methods can be used to validate an autonomous driving system (see **Fig. 5**). Test drives with test vehicles and AD prototypes can be used to test all vehicle components under realistic conditions. However, test drives are cost intensive. Moreover, critical traffic situations are rarely observed on the real road. Another possibility for validation is environment simulation. With synthetically generated data from high-fidelity simulators (e.g., dSPACE ASM), hazardous situations can easily be simulated and manipulated. For example, it is possible to change the weather and the time of day of a test for the same operational design domain. Although the quality of sensor simulations is steadily increasing, synthetic data will never truly match the real world due to the simulation paradox.

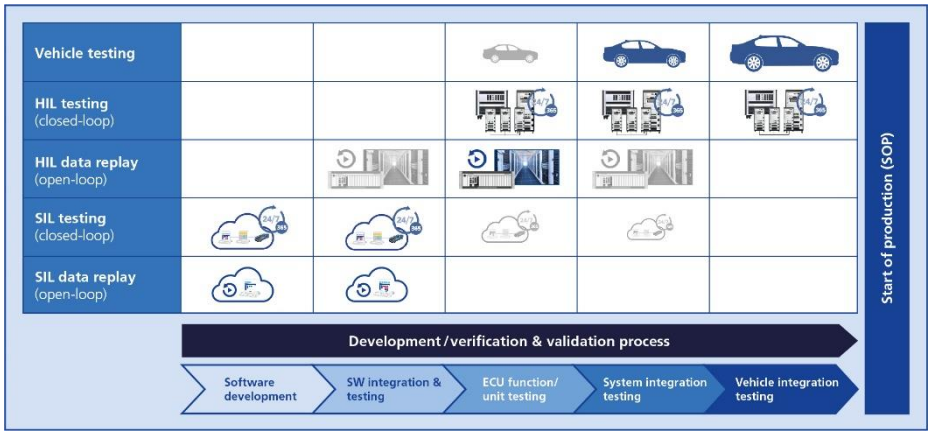

Fig. 5. Applicability analysis of different tests according to “safety first” for automated driving.

Therefore, replaying recorded sensor data, known as data replay (i.e., data reprocessing), has established itself as another key test methodology. Data that was previously recorded during test drives is used for validation. When replaying the data, the recording conditions must be recreated exactly as recorded, thus allowing for street conditions to be replicated in the laboratory. When using real-time data replay, the environment perception algorithms are supplied with recorded data around the clock, the system responses are measured, and the object detection quality is evaluated continuously. This ensures that the new software version of the AD stack is tested properly against real-world data and scenarios.

When testing with recorded and/or simulated data, it is possible to distinguish between two basic test methods. Software-based validation such as software replay focuses on validating an algorithm without considering time constraints or connected bus systems. Testing can be performed before a hardware prototype is available. The second level is hardware-based validation such as hardware replay, which involves deploying the central computer or sensor ECU, which is connected to the test system and supplied with synthetic and/or real data. This makes it possible to test all of the hardware as well as the electrical interfaces and the software in the conditions closest to on-road testing.

An example architecture (see **Fig. 6**) can be used to better understand the challenges of data-replay. In this architecture, the sensors of the autonomous vehicle generate a data volume of 70 Gbit/s. The data replay test station must stream this 70 Gbit/s around the clock while ensuring that the device under test does not recognize that it is not installed in a real vehicle and driving on the road. The challenge lies in the nature of the streamed bits. It should be noted that raw camera data, radar data, lidar data, and especially data from CAN and Ethernet (SOME/IP) packets are extremely different. Nevertheless, all these data streams must be played back in a synchronized manner within microseconds to ensure the correct test conditions. Taking the software replay scenario into account, an additional layer of requirements is added: the correct simulation of the virtual image of the device under test. Not only does this need to be accurate, but it must also contain the accurate simulation of the virtual buses attached to this virtual device under test. In addition, the whole software replay test has to run as fast as possible, ideally faster than in real-time.

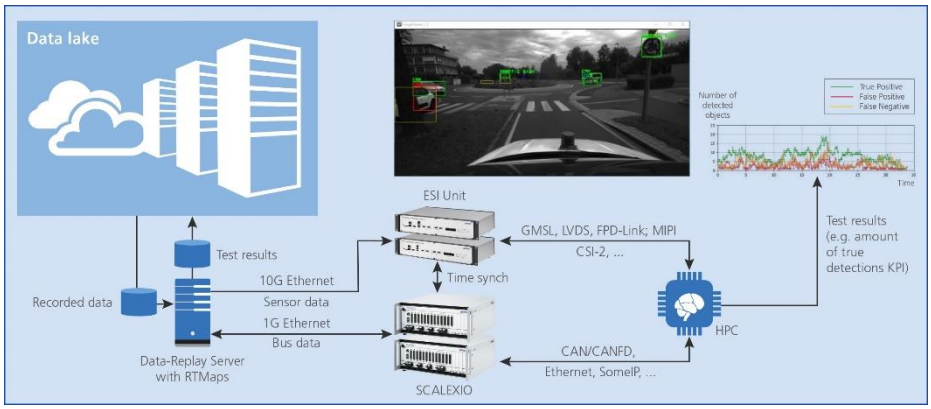

Fig. 6. An example architecture of a hardware replay test station

Data replay methodology can be enhanced even further. Introducing failures and manipulating data in the streamed data makes multiple test variants possible. The real-time manipulation of the streamed bus/network data makes it possible to validate the bus security stack of the device under test. Moreover, manipulating sensor data and

introducing failures can test the reaction of the perception algorithm against vandalism attacks.

Nevertheless, the data replay test station, be it software or hardware, is part of a bigger data and software delivery pipeline. Continuous testing and over-the-air software updates mean the faster a test campaign is carried out the faster car manufacturers can monetize new features. A crucial factor for this is the transfer of data from the data lake to the data replay test station. An optimal approach here is to bring the data replay test stations close to the measurement data, thus saving time and money. This is more challenging for hardware data replay systems than for pure software data replay systems with regard to colocation with the data storage.

With such an approach, the environment perception components can be tested 24/7 against thousands of driven kilometers. Being infrastructure agnostic is automatically required to pursue this endeavor as data could be stored in a native public cloud, on-site, or in hybrid cloud/edge constellations. Especially for hybrid cloud constellations, there is a great need for a single data access point to manage and allow for data replay test campaigns as it simplifies the testing process significantly. Intempora Validation Suite (IVS) abstracts these different data infrastructures and provides a single web interface to control both software and hardware data replay test stations.

Data replay is becoming increasingly established as a centerpiece of autonomous driving homologation as it guarantees a high level of confidence in the vehicle's perception stack. Nevertheless, data replay testing poses challenges regarding the data replay test bench due to the large number of sensor and bus interfaces and with regard to the infrastructure and efficient data storage [8].

A smooth transition between software and hardware data replay testing accelerates the entire validation process. However, the increasingly common use of public cloud services complicates the situation. The public cloud infrastructure is inaccessible to test engineers and the data must be co-located using the device under test. A new level of cooperation is needed among automotive OEMs, tooling providers, and the IT companies providing the infrastructure (**Fig. 7**).

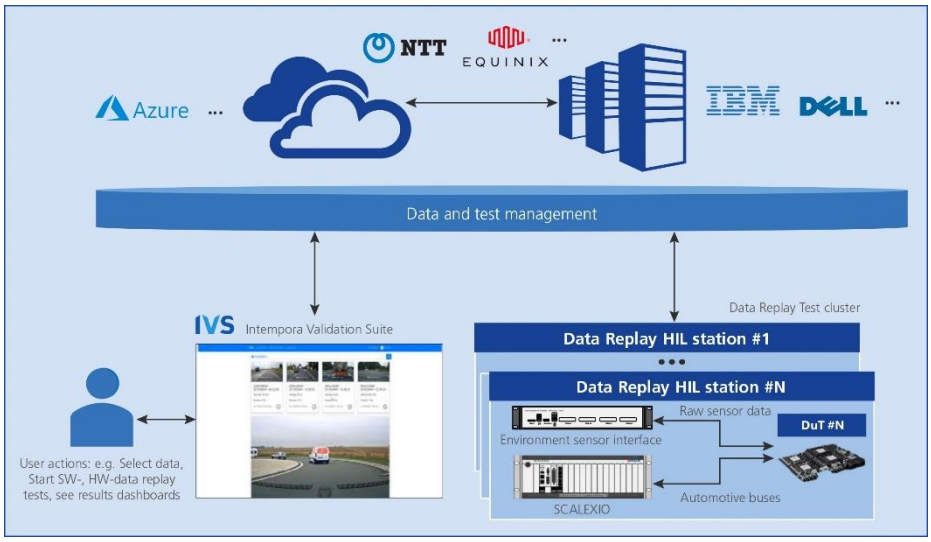

Fig. 7. Scalable data replay from dSPACE in the cloud and colocation center

6 Data Beyond Reality

Validating ADAS or AD systems with recorded data from real traffic situations is not enough. The real world is very complex, traffic situations are varied, and the validation timeline is limited. The vehicles must be prepared for situations that were not encountered during the data collection or testing phase. This is where simulation comes into play. The realistic simulation of sensors and the optimization of the total test cases are major challenges.

Regardless of the challenges, the need for synthetic data is evident. Therefore, well- defined data pipelines are able to seamlessly employ both real and synthetic data during the development process. As synthetic data is becoming more and more realistic, both data sources can be used for training and validation. Data management systems must be able to distinguish between these two types of data and maintain the connection if synthetic data is derived directly or indirectly from the collected data.

There is also a new approach to how to expand the data sources for the training and validation used in data management and the data pipeline. This is referred to as data augmentation and it is a combination of real collected data and synthetically generated data. There are two ways to augment a dataset. One is to replicate real data in the simulated environment with additional traffic participants. This becomes a fully simulated output for the system under test. The second option is to integrate a simulated object in the collected data. Most of the output is still collected data (the exact data from the sensors recorded), but that data is overlaid with artificially created objects or effects.

7 Closing the Loop

It is indisputable that data from real traffic situations plays a key role in the development of new cars and will continue to do so in the years to come. ADAS functions are mandated by regulatory authorities and there is no way around this. Even with basic ADAS functions, machine learning models are employed. With the complexity of the functions increasing and vehicle autonomy growing, more and more neural network models and their parameters are being integrated into the car software. Working efficiently with data and not just with code increments is a challenging task for companies that are not prepared, meaning they do not have proper infrastructure and have not adopted adequate methods to progress as expected by the market. Many companies have designed their internal tools to support data-driven development. However, it is evident that such approaches are difficult to scale, in terms of the data volumes, team size, and new function development. Internal tool development binds resources and keeps companies from achieving their main goal. Fortunately, commercial solutions have emerged to help build an automated, flexible, and scalable pipeline with stable features and secure maintenance to keep the pipeline running during the critical phase of development projects.

References

1. BMWGroup. [Online]. Available: https://www.press.bmwgroup.com/global/article/detail/T0293764EN/the- new-bmw-group-high-performance-d3-platform-data-driven-development- for-autonomous-driving?language=en; last accessed 2020/04/08. [Accessed: April 19, 2021].
2. A. Karpathy, "Software 2.0," November 11, 2017. [Online]. Available: https://karpathy.medium.com/software-2-0-a64152b37c35. [Accessed: April 8, 2021].
3. Ambarella, "Ambarella introduces CV5 high performance AI vision processor for single 8K and multi-imager AI cameras," January 11, 2021. [Online]. Available: https://www.ambarella.com/news/ambarella-introduces- cv5-high-performance-ai-vision-processor-for-single-8k-and-multi-imager- ai-cameras/.
4. Microsoft, "Microsoft docs," [Online]. Available: https://docs.microsoft.com/en-us/azure/expressroute/expressroute-erdirect-about. [Accessed: May 9, 2021].
5. P. Moravek, "Smart Data Logging – Part I: Reducing Redundancy," dSPACE, February 15, 2021. [Online]. Available: https://www.dspace.com/en/inc/home/news/engineers-insights/smart-data-logging-redundancy.cfm. [Accessed: May 5, 2021].
6. D. Hansenklever, "Dataset Enrichment Leveraging Contrastive Learning," December 7, 2020. [Online]. Available: https://towardsdatascience.com/dataset-enrichment-leveraging-contrastive- learning-ea399901f24. [Accessed: May 2, 2021]
7. M. Mengler, "Quality — The Next Frontier for Training and Validation Data," May 17, 2018. [Online]. Available: https://understand.ai/blog/annotation/machine-learning/autonomous-driving/2018/05/17/quality-training-and-validation-data.html. [Accessed: May 13, 2021].
8. dSPACE, "Validating ADAS/AD components using recorded real-world data," [Online]. Available: https://www.dspace.com/en/inc/home/applicationfields/our_solutions_for/driver_assistanc

e_systems/data_driven_development/data_replay.cfm#179_55577. [Accessed: April 20, 2021].