PRISSMA Project
Plateforme de Recherche et d'Investissement pour la Sûreté
et la Sécurité de la Mobilité Autonome
04/2021 - 04/2024

[L7.3] MISE A JOUR CONTROLEE DES ELEMENTS DU DOSSIER (DONT LES SCENARIOS ET LES OUTILS)

Main authors : C Gava (Sphéréa) and M Kaczmarek (Airbus Protect)

Keywords: Maintenance, MOC, Life cycle management, Diagnosis, AI neuron network diagnosis, accident …

Abstract. During the operation and maintenance of a vehicle or a fleet of automated or autonomous shuttles, unforeseen events in the design and engineering process may occur and cause an unexpected situation or behavior of the vehicle, or even worse an accident. Depending on the identified root cause of this unwanted event, either AI directly contributed (as did some attached equipment) or the cause is explainable by conventional systems engineering. It is this new paradigm that all players in the system life cycle must now adopt, in order to make the relevant decisions with the appropriate means of action and maintenance.

Résumé. Lors des opérations d'exploitation et de maintenance d'un véhicule ou d'une flotte de navettes automatisées ou autonomes, des évènements imprévus lors des phases de conception peuvent se produire et provoquer une situation ou un comportement inattendu du véhicule voire pire un accident de circulation. En fonction de la cause racine identifiée de cet événement indésirable, soit l'IA a directement contribué (ainsi que les équipements rattachés) soit la cause est explicable par l'ingénierie système conventionnelle. C'est ce nouveau paradigme que doivent maintenant s'approprier tous les acteurs du cycle de vie du système, afin de prendre les décisions pertinentes avec les moyens d'action et de maintenance appropriés.

1. INTRODUCTION : LIFE CYCLE MANAGEMENT FOR SYSTEMS INTEGRATING AI BASED TECHNOLOGIES

Performing Life Cycle Management of a system integrating AI based modules is a real challenge, considering specific properties of AI technologies. Conventional engineering systems and tools (analysis MBSA and engineering MBSE) are the 'traditional' activities of a physical and mechanical system which also address for example an automatic or autonomous vehicles.

This is obviously the case for automatic shuttles, and the most important feature of the Life Cycle Management requirements has to cover relevant feedback and corrective action when an unacceptable operational situation has been experienced in the operational cycle of the system.

The use of AI technologies and compatible devices and equipment leads to demonstrate their level of safety with new appropriate methods. At every stage of the W shape life cycle scheme, the challenge is to use the most relevant tools.

To illustrate this concept, in the case of an unacceptable behavior, one has first to identify the single cause or multiple causes or root cause in one or more steps of the life cycle development and deployment steps, and then to setup proper corrections.

2. CONCEPTS

Different kinds of corrections have to be envisaged, depending on the nature of the causes diagnosed:

- If one cause is a failure mode of a hardware component or mechanical module, a proper corrective maintenance task can be enforced, in accordance and compliancy with the maintenance policy of the system: this failure mode refers to an identified Line Replaceable Unit which can be exchanged on site, or on another maintenance level, regarding the maintenance concept

- If one of the possible causes is a software error, not relating to an AI technology, a cause analysis has to be applied to the software: it can be a specification error, or a coding error, and in both cases update of the software may be in question, as well as to find out why in the development process this error has been let unknown or not anticipated.

- If one of the possible causes addresses an AI based software or an AI brick embedded on a device, a cause analysis has to be applied to the AI architecture. The state of the art concerning this essential topic will be described in § 3.5. The challenge is to control this reparation but not only the removal of the inappropriate effect faced during the identified driving context. A global check and demonstration of no regression of the global AI functions and safety level has to be validated.

3. ARTS MOC PROCESS

Previous considerations make us structure the second part of ARTS Life Cycle, which is Operation and Maintenance phase, what we summarize "MOC" ("Maintien en Condition Opérationnelle" which means Maintain in Operational Condition).

The aim of this document is to suggest how the global process of MOC can be adapted to the specific context of automated and autonomous shuttles, and to provide basis recommendations to all the actors from the operators to the maintainers of the system.

The steps of the MOC process are classified the following way:

Process 1: Process 1: Shutlle supervision and associated data collection

(surveillance des navettes et recueil des informations associées)

Process 2: fault detection. Irrelevant behaviours identification (identification des comportements non prévus)

Process 3: non acceptable behaviours identification (qualification des comportements non acceptables)

Process 4: non acceptable behaviours diagnosis (diagnostic des comportements non acceptables)

Process 5: diagnosis of AI bricks (if involved) diagnostic des briques AI impliquées(si besoin)

Process 6: implementation of correctives actions (mise en œuvre des actions correctrices)

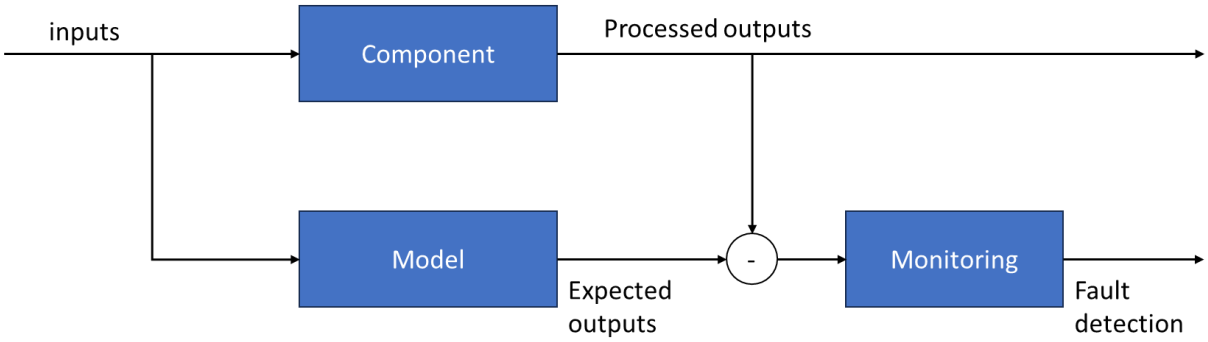
Process 7: validation of correctives actions (validation des actions correctrices (non régression, analyse d'impact))

Each one of these processes will be described on two aspects, the conventional technics of maintenance and MCO and their necessary evolutions to highlight the new AI constraints and to anticipate basis recommendations. Managing global maintenance on transportation systems embedding technologies and monitoring systems based on AI, needs to guide and advise the operating and maintenance entities in their activities and decisions.

3.1. PROCESS 1: SHUTLLE SUPERVISION AND ASSOCIATED DATA COLLECTION

3.1.1. INTRODUCTION: STATE OF THE ART

The fault detection process is based on the following principles:



A component processes input signals to generate output signals. These outputs are compared to a reference from the engineering system analysis and then a logic decision (fault detected or not detected) is confirmed.

Data acquisition is the first part of the process. It consists to acquire data from an analogic signal or a numeric bus and to store it for future use in at least the monitoring process.

Challenges of acquisition process:

- The first one is to know which information has to be caught. Only a testability analysis, in other word, a study of the system, all its defects and impact of tests can determine the different options available to capture the relevant data.

- The second challenge is to capture the data. For analogic signals, sampling rate is important. Lower sampling rates can lead to wrong interpretation of the signals. At the contrary, higher sample rate requests more powerful capture device and needs more storage.

For numeric signals, the capture of the data shall no be intrusive (no signal alteration) nor induce delay in the transmission. Data consistency shall also be done to ensure that the captured data is the real one. Capture can consume lot of storage but several technics exists to reduce it. Some data can be very difficult to store, such as video.

Once data is acquired, it can be used for the monitoring process. The main idea is to compare the acquired signals with a reference and then produce a logic answer, in other word, the real signal is compared to the expected one, the system is operational, otherwise the system is faulty and an error is raised.

To do so, two technics exist for the comparison :

- reference data base

- comparison with a theoritical model

The first option consists to have a reference database where all expected situation are defined. The main advantage is that the comparison can be done very quickly and does not require a

powerful device to monitor the system. The main disadvantage is that all expected result shall be computed and defined during the design of the monitoring system.

The second option can deal with more situation. The main advantage is that monitoring can be done in real time, but also requires more powerful devices to compute calculation before performing the comparison. It also requires a model of what the component shall process in order to perform the comparison.

The monitoring process can be done in two ways:

- In real time
- In delayed time

Generally, real time monitoring is done on board, and is a part of the system it-self. It requires specific design to communicate with all other components (or at least with monitored ones and also shall be analysed to ensure that the monitoring component does not affect the safety of global system. It also requires a data storage space and be powered by the systems.

Offline monitoring is done off board and not in real time. It can be done during downtime or during specific period. It requires only collection and storage components as well as a transfer mechanism for the off-board monitoring systems.

On board and real time monitoring components are generally used for critical features.

3.1.2. IMPACT AND ISSUES OF THE AI EMBEDDED TECHNOLOGIES

For automated and autonomous shuttle, monitoring can be done both online and offline mode:

- online and real-time mode for ~~e~~-safety and security data from the digital infrastructure
- off-line mode for :
  - fleet management (maintenance policy and parameters update)
  - near accident and irrelevant behaviour investigation

The last point is the most important. Each near accident must be investigated to determinate the root cause of the event.

For an accident event analysis, the first decision of standardization is to set which parameters and external information (scene capture) are considered as evidences and must be stored for legal investigation.

In addition, acquisition duration is also an open question. Storage of data can be significant, especially for autonomous vehicle. Consistency of the data is a challenge, a timestamp shall be added to all gathered signals. More question are also open, such as the ability to keep the data safely during an accident. Which mechanical constraint shall be observed to guarantee the data integrity? Privacy is also a challenge. Data such as personal information or relative to the person (such as geo data) are sensible and shall not be shared to third party.

3.2. PROCESS 2: FAULT DETECTION. IRRELEVANT BEHAVIOURS IDENTIFICATION

3.2.1. INTRODUCTION: STATE OF THE ART

The fault detection process consists to identify if a fault occurred in the system. A fault is defined as the manifestation of an error on an item or system that may lead to a failure (ARP 4754 Rev. A). A failure is then defined as an occurrence which affects the operation of a component, part or element such that it can no longer function S as intended (this includes both loss of function and malfunction).

This capability leads to reduce diagnostic time or downtime in general and, therefore, increased system availability. A good inherent diagnostic of a system also enhances the crew-members' confidence in operating the system, the main driver of mission success (Fault-Detection, Fault-Isolation and Recovery (FDIR) Techniques, lesson 839, NASA).

Initiated in the early 1970s, these technics has been developed remarkably since then. Today, there are fully integrated into vehicle control systems, robots, transport systems, power systems, defence systems and so many other industrial applications.

The basic concepts behind fault detection is to perform observations of system behaviour which are not consistent with expected behaviour. This requires two important concepts, an observer and a comparison unit. If the observation given by the observer cannot be compared to the expected observation, a fault is detected.

These two concepts are the key points of fault detection for any system. It assumes that a fault can be observed, in other words, we are able to perform a measure. It means that a measurement location is defined and a mean to make the measure is also available. It also assumes that a comparison can be done between the measure and the expected result. At least, a decision is made after the comparison. The system is healthy or is faulty.

When automated system or human operated systems are considered, the expected results can be defined from specifications and mission goals. But when system considered are autonomous vehicle, definition of an expected behaviour and an anormal situation can be difficult.

Thus, the notion of a near-accident arises. This notion is used to define a not acceptable situation for the autonomous vehicle without accident (i.e. collision). Such situations are important to be characterized in order to detect a new kind of fault for autonomous vehicle, a situation computed by the AI which leads the system to a faulty situation.

In addition, a worst scenario can occur for any system, the latent failure. In this case, a fault occurs in the system but is not detected or has not immediate effect. Once conditions are met, the failure arises which can lead to catastrophic events. Detection is then a crucial aspect for safer systems and multiple technics are used to reduce these risks.

To deal with these concepts, model based diagnostic engineering is a science starting in the 1970s to cope with poor reliability of systems and the demand for more complex and safe systems (Design for Testability, 2008, DSI International). In addition of RAMS study, testability engineering and diagnostic engineering use same models to perform analysis. Testability focus on how the system can be tested to detect fault (for a given set of defined and modelized fault in the system, does the system can be tested to find them ?) The important ratio is the number of detected failures versus the total number of failures. This metric is called "fault detection".

Fault detection can be done manually or automatically, depending of the complexity of the system and how quickly the system shall be recovered. Generally, HMI are subject to be visually and aurally controlled by human (quick response of operator), but for embbeded system, the most popular technics used is the built-in test (BIT). BIT system can vary from a simple LED to a complex computation to determine the health status of a component. All information are generally centralized into the Health and Usage Monitoring System (HUMS) which gathers and computes a global health metric and sometimes try to estimate the remaining useful life of the system (RUL) to determinate if the system can perform its mission or not.

3.2.2. IMPACT AND ISSUES OF THE AI

For an autonomous vehicle the fault detection should be mandatory for safety reason. Failures which can lead to dangerous situation shall be detected. It is also a requirement to manage a fleet and not to leave a faulty vehicle in operation.

The fault detection is also important to determinate if a function processed by an AI component is not working properly.

3.2.3. DIAGNOSIS OF THE AI BASED SOFTWARE

This task may be a tricky task taking into account inherent properties of AI technologies and scientifical domain which evolutes pretty fast.

These contributions should be qualified in the real world, as trustworthiness of models supporting simulations which remain currently partial: replicability and repeatability of the unacceptable situation to which AI component has contributed. They would be decisive about the fact to qualify the unrelevant behavior and internal diagnosis.

3.3. PROCESS 3: NON ACCEPTABLE BEHAVIOURS IDENTIFICATION

3.3.1. INTRODUCTION : STATE OF THE ART

Qualification of the fault can be done once the faulty item is identified and the failure mode is known. The associated severity has an important impact with the maintenance process or the recovery mode to use.

The qualification and associated actions shall be planned during the design phases. It exists three main recovery modes, (a) full recovery using redundant system components, (b) full recovery using alternative path and (c) degraded mode.

For category (a) the system is designed to be resilient against fault and redundant component are available to compensate a faulty element. This is generally used for critical components where a fault can lead to the destruction of the system and injuries or worst death of people.

For category (b), the design used alternative component to achieve the expected mission. Generally, the systems have some limitations.

Category (c) means generally the loss of the functionality. For critical systems, it can affect the achievement of the intended mission but does not lead to critical events (for aircraft, a category (c) failure can be the lost the entertainment system during a flight, or for a vehicle the loss the radio. In both cases, it could be annoying for people but does not affect their security).

For autonomous vehicle, this part is crucial. The qualification is mandatory to determinate if the fault is due to a non-AI component or is due to AI component. In the first case, classical technics can be used to manage the failure. But for IA components, new technics shall be developed to capture the global situations (ie. all relevant data) which has led to the unwanted situation (near accident or accident) in order to determinate what are the root causes of the failure.

For AI components, three categories can be defined:

1. AI environment
2. AI training
3. AI core

For the first element, the AI didn't response in time or the expected solution due to runtime environment (memory leak, time response, ...). For the second one, the situation encountered was not defined in the training set during the design of the AI. Then, the third one corresponds to a wrong answer of the AI from a known situation.

3.3.2. IMPACT AND ISSUES OF THE AI

Fault qualification is crucial to determine the recovery process (if needed or required) and the adapted maintenance.

In case of systems supporting AI bricks, the Diagnosis consists to analyse the whole System Engineering Framework.

Many components or layers of System Engineering Framework may be also addressed to explain why the use case under analysis where system has failed, has not been clearly identified, and has not been therefore anticipated in the design and development process, which implies that this process has to be updated in a way, as is illustrated by following questions:

- Have requirements been sufficiently formulated? May be failed use case was not covered by these requirements…
- Have the ODD been correctly determined? May be failed use case went beyond the ODD…
- Have OEDR been correctly formulated? May be failed use case were not integrated in the OEDR…
- Which library of scenarios and use cases have been simulated in the virtual testing campaign? May be failed use case has been forgotten in this library…
- Which families of scenarios and use cases have been simulated in the controlled testing campaign? May be failed use case has been forgotten in these families…
- Which families of scenarios and use cases have been simulated in the real environment testing campaign? May be failed use case has been forgotten in these families…
- Which metrics have been applied in post processing of virtual, controlled or real testing campaigns? May be failed use case had been assessed with a not convenient metric
- Which criteria had been applied for the characterization of the near miss accidents? May be these criteria were too optimistic…
- Which platform was used for the simulation of the failed use case? May be this platform had led to optimistic results

3.4. PROCESS 4: DIAGNOSTIC OF FAULTY ITEM (ISOLATION)

3.4.1. INTRODUCTION: STATE OF THE ART

The next step after the detection of the fault is to find which component is faulty. This step has a direct impact for the maintainability or the level of service of a system. Hence, a system which is able to detect and find which component is faulty can (if possible) manages a recovery mode and then continues its mission. In case the system cannot be in a recovery mode, the downtime to identify the failure and then to replace the item can be reduced.

One important aspect in the isolation process is the fault group item (called fault group). It represents the group of components which could be faulty but we cannot know exactly which of them has a failure. In the best case, the fault group size, in other words the number of items contained in the fault group, is equal to 1, which means that the item can be fully isolated if it fails.

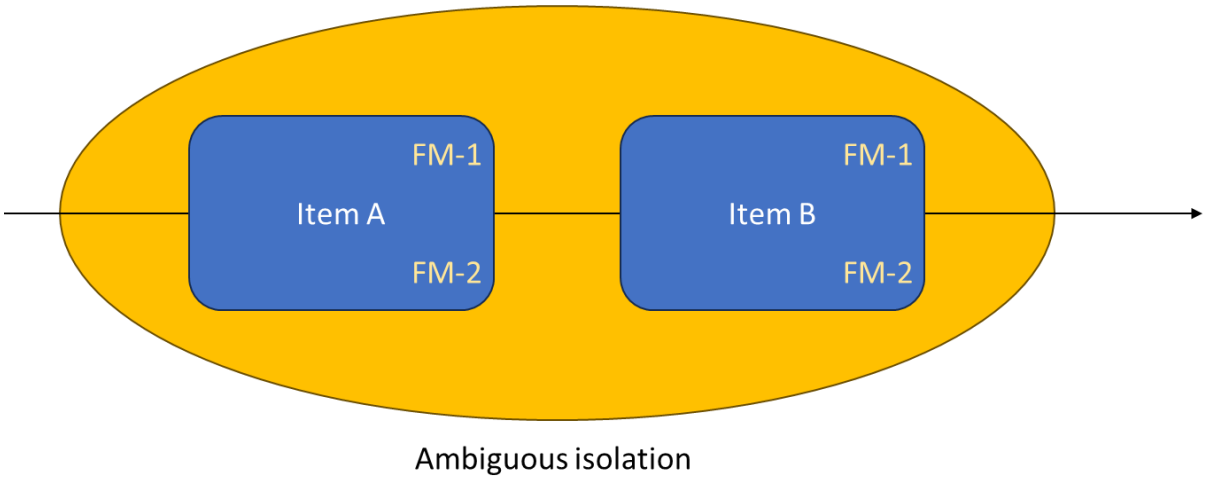


Figure 1 Ambiguous isolation - fault group size equal to 2

In the figure above, we have the representation of a functional chain with two items. At the end of the functional chain, we can detect a fault but we are not able to determine which item is faulty and must be replaced. In this case we call this situation an ambiguous isolation. The impact in term of maintenance is that both items shall be replaced, and then cost and time for maintenance increase.

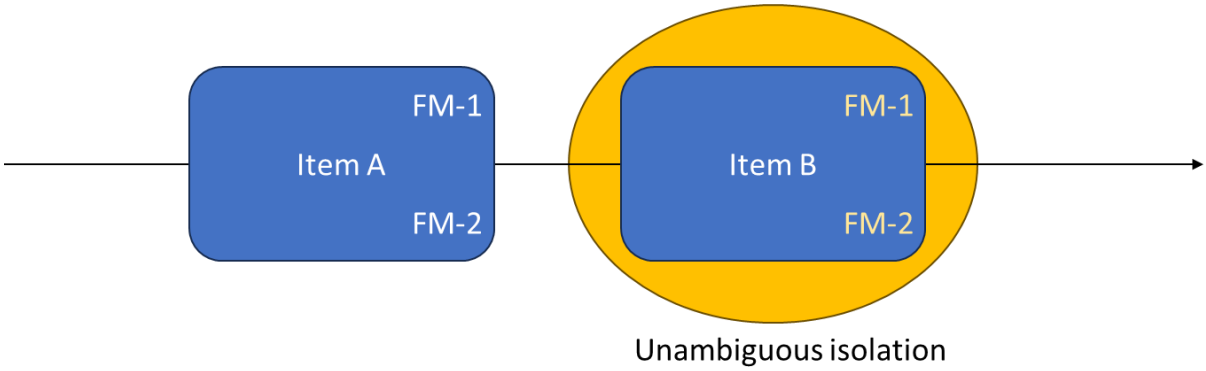


Figure 2 Unambiguous isolation

In the figure above, the fault group size is equal to 1. We are able to discriminate which item shall be replaced. Maintenance is then enhanced as only one item have to be replaced. Cost and time are then reduced.

The process to have a set of tests to detect and isolate fault is called diagnostic. Generally, a fault tree localization is created or generated (express, DSI International). Its execution is called 'perform a diagnostic'. As the detection, it could be done manually or automatically.

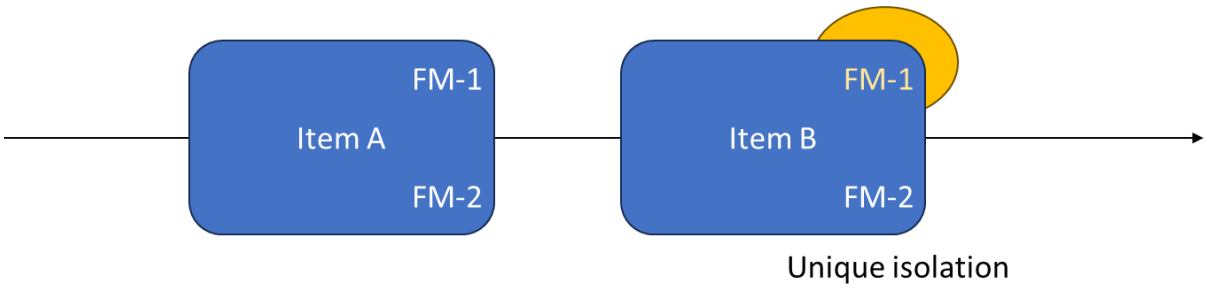


Figure 3 Unique isolation

Figure above shows a more precise process of diagnostic. Instead of stopping the isolation at the faulty group or the item, diagnostic goes further and tries to identify the failure mode associated to the detected fault. This step is crucial for the last step of the fault detection and isolation process, the qualification of the fault.

3.4.2. IMPACT AND ISSUES OF THE AI

Isolation is required for autonomous vehicle to discriminate or incriminate the AI component and then to adjust the recovery and maintenance process. The diagnosis process is similar to the one described previously for irrelevant behaviours.

3.5. PROCESS 5: DIAGNOSIS OF AI BRICKS (IF INVOLVED)

3.5.1. DIAGNOSIS OF THE AI BASED SOFTWARE

This task may be a tricky task taking into account inherent properties of AI technologies and scientifical domain.

These contributions should be qualified in the real world, as trustworthiness of models supporting simulations, remains currently partial: replicability and repeatability of the unacceptable situation to which AI component has contributed would be decisive about the fact to qualify the unrelevant behaviour and internal diagnosis.

Every single change in an AI brick needs to validate the non-regression of the corrective action or process.

Corrective action on a faulty software has to remove a faulty behavior, but at the same time, one has to be sure that it does not produce additional miss behavior on other use cases, which were not failing before. This a tricky issue which is not yet wholly covered by the state of the art but in which alternative solutions are proposed.

3.6. PROCESS 6: IMPLEMENTATION OF CORRECTIVES ACTIONS

3.6.1. INTRODUCTION: STATE OF THE ART

The corrective maintenance takes place once the fault is detected, isolated and qualified. Two cases for this step :

1. Classical corrective maintenance
2. AI corrective maintenance

The first case happens where the fault occurs on a non-IA component of the autonomous vehicle. A classical maintenance process shall be applied to restore the system into its nominal state.

The second states need to be defined. In this case, the qualification states about three sub cases (environment, training and core). Depending of the sub-case, the corrective maintenance will differ.

In the case, the AI environment is faulty, corrective maintenance shall restore the runtime environment in a state where AI runs as intended.

In the case the training set lacks of a specific situation, a new version of AI can be developed to take into account this situation. Then this new version can be deployed to the whole fleet of autonomous vehicle, or to the full set of similar vehicles. Please note that the situation shall be analysed to determinate if the correction shall be:

- nothing, as the situation is very unlikely (acceptable risk)
- non-AI workaround to deal with this situation
- new version of AI component

In the case of a bug in the core of the AI component. specific technics currently in development shall be used to find the internal root cause of the issue and then try to find a fix (creation of new upgrade of the AI component)

The main concern when a new version of an AI component is required to fix the failure is to not introduce a regression and then increase the risk of near accident or accident.

3.6.2. IMPACT AND ISSUES OF THE AI

The corrective maintenance for AI component is a new field of research and is the new challenge for the coming years.

3.6.3. CORRECTION OF THE AI BASED SOFTWARE

To find proper correction of AI based software able to re-establish convenient and acceptable behavior of the whole system in the use case addressed originally, one has to conduct a deep survey to identify part of the software to correct and precise elements to change, update or remove.

For example, if AI software is based on Neural Networks, one has to find out what layer(s) of the networks to modify, and what value of weights to modify and readjust to obtain a proper correction of the global behavior of the top level system in addressed use case.

In opposition with non AI diagnosis tools, there is not a large panel of relevant methodologies and tools available to diagnose AI bricks and systems.

The learning models of the AI bricks of the autonomous driving system require diagnosis when failure cases are encountered during the operation of the autonomous vehicle. These learning models have to follow an elaborate testing and certification process to avoid accidents. This process is time consuming and can take up to 6 months to 1 year for each update. However, we expect that customers will always encounter failures that are underrepresented in the training data and not taken into account in the test data or due to missing features in the learning model.

Thus, an important issue facing autonomous vehicle operators is the maintenance of the autonomous driving system software of AI bricks between major software updates, in order to fix the driving behaviour of the autonomous module on the encountered failure cases or to add the requested missing functionalities of the model without the need to validate the whole system from the beginning. We believe that the diagnosis and maintainability of learning models are important challenges for the success of autonomous shuttles. The maintainability of autonomous driving systems must correct the failures of the learning models without changing the driving behaviour over all the kilometers that have been successfully driven before.

Example of corrective maintenance techniques for AI DNN based bricks using deep networks.

DNNs have been successfully applied to a wide variety of problems, including image recognition, medical diagnosis and self-driving cars. However, DNNs are far from infallible, and bugs in their algorithms have led to loss of life and unjustified arrests. This has motivated recent advances in understanding, verifying, testing and fixing DNNs.

After deployment, the DNN may have errors, for example due to misclassification of images. To repair the DNN, one could re-train the network using the original training dataset with the newly identified faulty inputs added. However, retraining is extremely inefficient (ie. training can take a long time (days or weeks with the most sophisticated equipment). Even worse, the original training data set may not be available for retraining; for example, it may be missing or lost. Indeed, in some cases, privacy regulations require companies to regularly delete private data. Retraining can also lead to arbitrary changes in the network and often introduce new bugs in the behaviour of the network. In addition, standard software programs and DNN models are fundamentally different in terms of bug detection and correction. For example, standard software programs are tested by comparing the actual output with the expected output. If the two outputs are different, then we consider the program to have a bug. On the other hand, DNN-based software has a complex structure, and it learns from a set of training data. If the DNN produces an incorrect classification during training, we call it a failure case, it is not necessarily that the DNN contains a bug, because a DNN cannot guarantee 100% correct

classifications. Furthermore, un-like an ordinary program, a DNN program uses weights between neurons and non-linear activation functions for similar purposes, which makes debugging and testing DNN-based software difficult [32]. Several researchers have been interested in this problem and have proposed approaches and tools for diagnosing and correcting bugs in deep learning programs. In the following section we present some of them.

3.6.4. UPDATED FEEDBACK ON BUG DIAGNOSIS TOOLS FOR DEEP NEURAL NETWORKS

DeepFD: Automated Fault Diagnosis and Localization for Deep Learning Programs. DeepFD is a learning-based fault diagnosis and location framework that transforms the fault location task into a learning problem. It deduces suspected fault types by monitoring runtime features extracted during DNN model learning, and then locates faults diagnosed in DL programs. It overcomes the limitations by identifying the root causes of faults in DL programs instead of neurons, and by diagnosing faults using a learning approach instead of a set of hard-coded rules. The evaluation shows the potential of DeepFD. It correctly diagnoses 52% of faulty DL programs, compared to about half (27%) obtained by the best state-of-the-art work. Furthermore, for fault location, DeepFD also outperforms existing work, correctly locating 42% of faulty programs, almost doubling the best result (23%) obtained by existing work. Figure 4 shows the DeepFD workflow. It consists of three steps: (1) diagnostic feature extraction, (2) fault diagnosis and (3) fault location.

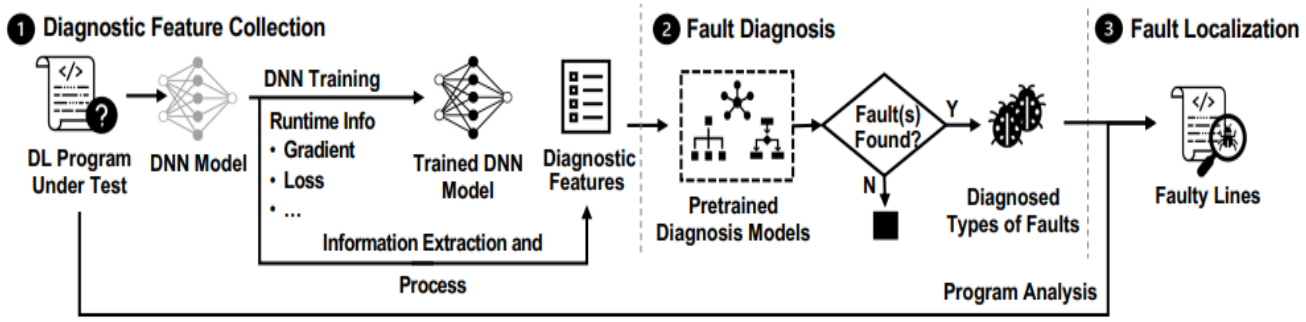


Figure 4: Overview of DeepFD [32]

Step 1: Extraction of diagnostic features. Given a program, DeepFD builds a DNN architecture and collects runtime data such as loss and information from neurons while training the architecture. However, storing all the weights and gradients of each neuron at each training iteration is expensive. Referring to recent work on dynamic trace collection and runtime monitoring, DeepFD collects 20 runtime data, including loss and accuracy, the number of times the loss increases, the average or standard derivation of weights, etc. The data is collected repetitively at certain intervals (eg. every 256 batches or every epoch). Finally, 160 diagnostic

features are extracted by applying statistical analyses (e.g. calculating variance and skewness) to the collected data.

Step 2: Fault diagnosis. After obtaining the diagnostic features, the possible types of faults that exist in the DL program are then deduced based on the features. A multi-label classification problem, which maps the obtained features to one or more possible labels, is considered. Each label corresponds to a type of defect. The classification is based on predictions made by a set of pre-trained diagnostic models. The diagnostic result is given by the union of the diagnosed faults predicted by each diagnostic model in order to maximize the number of diagnosed faults. In addition, for each DL program tested, we run it ten times to account for the randomness of the training of DNN models.

Step 3: Fault location. After the acquisition of the diagnosed fault types, DeepFD performs fault localization at the program level. Specifically, the program is first analyzed in its abstract syntax tree, and DeepFD goes through the nodes of the analysis tree, traverses the assignment statements as well as the expressions, and then identifies the locations (i.e. the lines) where the diagnosed fault types are defined. For example, to locate the optimizer in the source code, DeepFD looks for invocations in the source code. In the source code, DeepFD looks for invocations to specific pattern formation APIs (i.e., fit), and analyzes the argument and keywords of that node, and finally returns the line number where the optimizer is assigned. However, this process is not always as easy as the identification of keywords. For example, the learning rate, as a hyper parameter of the optimizer, is usually omitted in the program (i.e. the default learning rate will be used), which makes keyword identification infeasible. In this case, DeepFD locates the line where the optimizer is defined, while also reporting that the fault is in the learning rate type, which can provide more accurate debugging information to developers. Since a fault can involve multiple lines, DeepFD reports a set of suspect lines of code for each fault diagnosed.

DeepDiagnosis: Automatically Diagnosis Faults and Recommending Actionable Fixes in Deep Learning Programs. Descriptions of failure symptoms and their root causes are provided. The process of matching symptoms to their root causes is described. This approach monitors key values during training, such as weights and gradients. During training, it analyses the recorded values to detect symptoms and determine if a training problem exists. If a symptom is detected, DeepDiagnosis invokes a decision tree to diagnose/repair the information according to a set of predetermined rules. If not, the training terminates with the trained model and the model declared correct [35]. Figure 3 provides an overview of the DeepDiagnosis approach for locating faults and suggesting locations for repair. DeepDiagnosis receives as input the initial model architecture with a training dataset and passes the callback method as a parameter to the training function (fit()) (see Figure 5, left-hand side). This callback captures and stores key values (i.e. weight, gradient, etc.) during the forward

and reverse propagation steps (see Figure 3, middle section). Then, the DeepDiagnosis applies a dynamic detector during training to signal different symptoms at different stages depending on the error conditions. If the DeepDiagnosis detects a symptom, it analyses the recorded key values to determine the likely location of the input pattern for correction (see Figure 5, right-hand section). Finally, DeepDiagnosis reports the type of symptom, the layers and the stage in which the symptom was detected, and suggests a location for correction.

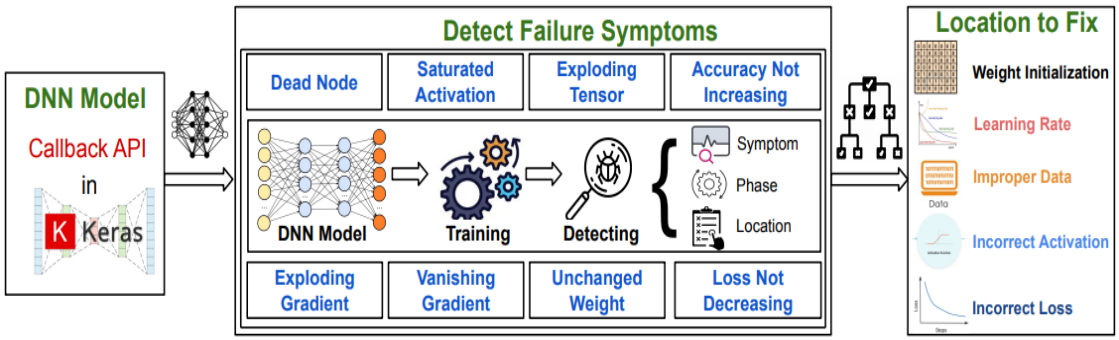


Figure 5: Overview of DeepDiagnosis [35]

3.6.5. TECHNIQUES FOR REPAIRING DEEP NEURAL NETWORKS

ARACHNE: Search Based Repair of Deep Neural Networks. Arachne is a DNN repair technique which locally modifies a DNN without re-learning by changing the model parameters (i.e. weights) in an exploratory manner. It works by identifying a weight that induces misclassification in the trained model, adjusting the value of the weight using the particle swarm optimization technique, and correcting the misclassification to obtain the expected classification. Arachne performs two main operations: localization and patch generation. In the localization phase, given a set of inputs that cause unexpected behavior, Arachne identifies a set of neural weights that are possibly related to the observed misbehavior. This phase is called localization; the intuition is that changing the values of these neural weights is likely to affect the behavior of the model, possibly in the right way. Trying to adjust all the neural weights of a DNN model for repair would be too costly, as even relatively simple DNNs are composed of thousands of weight parameters. To reduce the search space, Arachne only considers the neural weights connected to the final output layer (see Figure 6).

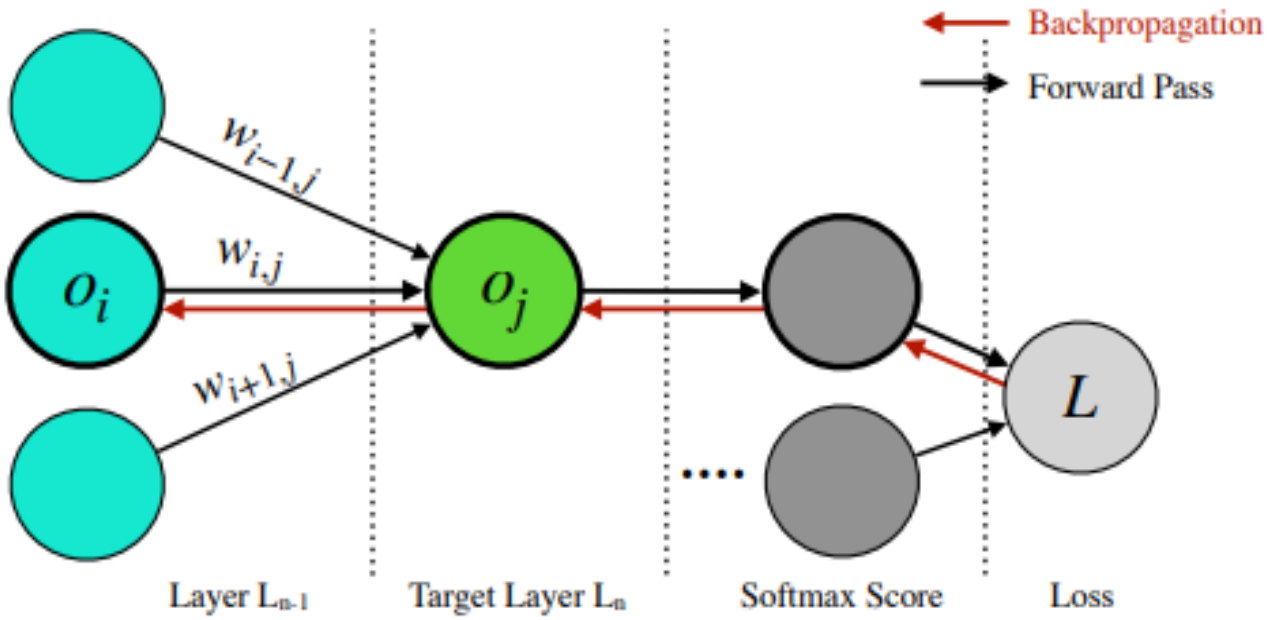


Figure 6: Neural layers and weights considered by Arachne [36]

In addition, Arachne follows a simple localization method to identify the weights that are likely to be responsible for the targeted misbehavior. The algorithm in Figure 7 presents the pseudo code of the localization method. First, Arachne sorts the neural weights according to the gradient loss of the back-propagation to the corresponding neuron (lines 1-6), and then considers only the first N_g neural weights to reduce the number of candidate weights (line 8).

In Figure 6, for the neural weight $w_{i,j}$ that connects the i^{th} neuron in layer L_{n-1} to the j^{th} neuron in our target layer, the gradient loss back-propagated from the final loss,L is calculated. In addition to the gradient loss, Arachne also considers the impact of each neuron weight on the final result, when back-propagating the faulty input (lines 8 to 12). The direct impact of a neural weight on the final result, fwd_imp is calculated simply as the product of the given weight and the activation value of the corresponding neuron in the previous layer: e.g., $o_i w_{i,j}$ for the neural weight $w_{i,j}$in Figure 4. This calculation is performed by ComputeForwardImpact (Line 10).

input : A DNN model to be repaired, **M**, a set of neural weights, W, a set of inputs that reveal the fault, I_{neg}, a loss function, $\mathcal{L}$, and the number of neural weight candidates to choose based on gradient loss, N_g
output: a set of neural weights to target for repair, W_t

```
candidates ← [];
for weight in W do
    grad_loss ← ComputeGradient(weight, M, I_neg, L);
    Add tuple (weight, grad_loss) to candidates ;
end
Sort candidates w.r.t. gradient loss;
pool ← {};
for i = 0 to N_g do
    (weight, grad_loss) ← candidates[i];
    fwd_imp ← ComputeForwardImpact(weight, M, I_neg);
    Add tuple (weight, grad_loss, fwd_imp) to pool;
end
W_t ← ExtractParetoFront(pool);
return W_t
```

Figure 7: Algorithm of the Arachne localization method [36]

In the patch generation phase, Arachne increases the negative inputs (i.e. inputs that lead to the unexpected behaviour) with a fixed number of positive inputs (i.e. inputs that are processed correctly). Similar to automatic program repair techniques by generation and validation (G&V) such as GenProg and Arja, positive inputs are used by Arachne's adaptation function to preserve the initially correct behaviour of the DNN being repaired. Following an adaptation function based on the execution of positive and negative inputs, Arachne uses Particle Swarm Optimization to search for the set of neural weights that would correct the behaviour of the DNN under repair.

Particle Swarm Optimization (PSO). PSO is used to repair DNNs by modifying their neural weights. A patch generated by Arachne is essentially a set of new neural weight values to be applied to the specified weights. Arachne represents this set of new values as a vector, which in turn represents the location of a particle in PSO.

PSO updates the localization vector,$\vec{x}$, at time step $t + 1$, using a velocity vector (equation 1). The velocity vector itself is calculated from the current velocity, $\overrightarrow{v_t}$, the location of the best fitness value the particle has observed locally so far, $\overrightarrow{p_l}$, and the location of the best fitness value the whole swarm has observed globally so far, , $\overrightarrow{p_g}$, each multiplied by uniform random bias vectors. The parameter χ is a constant multiplier calculated from the weights of the local and global components, ϕ_1 and ϕ_2 (equation 3): these parameters, collectively called "constriction coefficients", control the convergence of particles in a swarm without setting an explicit velocity limit. The same value for ϕ_1 and ϕ_2 is used.

$$\vec{x}_{t+1} \leftarrow \vec{x}_t + \vec{v}_{t+1} \quad (1)$$

$$\vec{v}_{t+1} \leftarrow \chi(\vec{v}_t + \vec{U}(0, \phi_1)(\vec{p}_l - \vec{x}_t) + \vec{U}(0, \phi_2)(\vec{p}_g - \vec{x}_t)) \quad (2)$$

$$\chi \leftarrow \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}, \text{ where } \phi = \phi_1 = \phi_2 \quad (3)$$

Initialization. PSO is a population-based global search algorithm, and requires that the location of each particle is initialized at the beginning. While uniform random initialization is a widely accepted method, Arachne attempts to repair an already trained DNN model and can benefit from the exploitation of learned weight values. Therefore, Arachne initializes the location of each particle according to the distribution of the learned weights. Each element of a particle vector is sampled from a normal distribution defined by the mean and standard deviation of all neural sibling weights. By sibling weights we mean all weights corresponding to the connections between our target layer and the one before it, i.e. all weights between $L_n$and L_{n-1} in Figure 4. The initial velocity of each particle is set to zero.

Fitness function. PSO updates each particle to the best-known location, which is determined by a fitness function. Like other G&V techniques, Arachne uses a fitness function which consists of two main parts: correction of unexpected behaviour and conservation of correct behaviour. The fitness function is defined as follows:

$$fitness = \frac{N_{patched} + 1}{Loss(I_{neg}) + 1} + \frac{N_{intact} + 1}{Loss(I_{pos}) + 1} \qquad (4)$$

The set I_{neg} contains the inputs that reveal the unexpected behaviour, while the set I_{pos} contains the inputs that are processed correctly: both sets are fixed by Arachne at the beginning of the repair attempt. Instead, $N_{patched}$ is the number of inputs in I_{neg} with output corrected by the current patch, while N_{intact} is the number of inputs in I_{neg} with output that is still correct. The model loss values obtained from these sets of inputs, $Loss(I_{neg})$ and $Loss(I_{pos})$, are used to reflect improvements and damages that do not correspond to changes in $N_{patched}$ and N_{intact}. Note that we add one to the numerators and denominators so that one can provide guidance when the other becomes zero, and vice versa.

Regression problem: Arachne identifies the parameters (weights) that affect the classification errors and searches for the weight values that reduce the error using particle swarm optimization. However, Arachne presents a potential risk of introducing new classification errors into the model while correcting some classification errors. The cause of this problem is that Arachne only uses misclassified data as information for fault location and does not take into account correctly classified data. In other words, if some of the weights identified by the fault location step affect correctly classified data, the patch generation step may turn correctly classified samples into misclassified samples. To solve this problem, TOKUI et al. made the following two assumptions: (1) It should be relatively easy to correct a misclassification if a misclassified sample has already been correctly classified during the training process. (2) If the weights that affect the results of correctly classified samples are identified, the regression of the data can be reduced by avoiding changing the values of these weights.

To avoid the problem of regression in DNN repair, TOKUI et al. proposed a new repair technique, called "NEURECOVER".

NEURECOVER: Regression-Controlled Repair of Deep Neural Networks with Training History. NEURECOVER, a new DNN repair technique that uses fault location with learning history. In this technique, fault location detects improved and regressed data in the learning process. The term improved data is used to refer to data that has moved from misclassification to correct classification and the term regressed data is used to refer to data that has moved from correct to misclassification. Then, as illustrated in Figure 8, NEURECOVER identifies weights that have changed significantly during the learning process and that do not affect the improved data but only the regressed data (Fault Localization Stage), and optimizes the localized weights by particle swarm optimization as in Arachne (Patch Generation Stage). The fault localization stage using the learning history is performed in the following three steps.

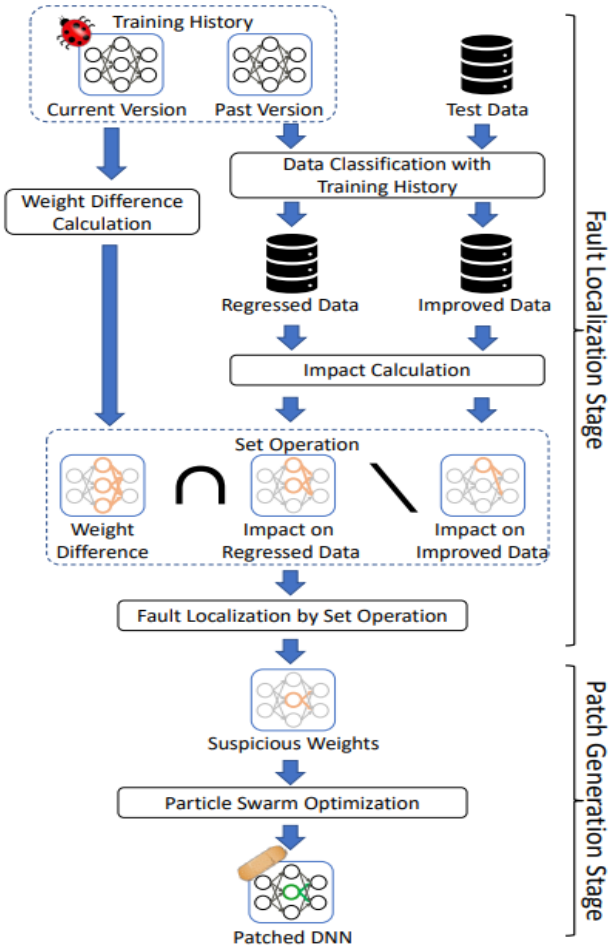


Figure 8: DNN repair technique [40]

Step 1: Data classification with training history. NEURECOVER detects regressed and improved data from the training history. To repair a model that has been trained for n episodes, it uses the weights of the models M_n and M_{n-k} in the training history, where M_i is the model that was trained during $i(1 \leq i \leq n)$ episodes. For each model, NEURECOVER examines the predictions of the data set. It classifies data that have changed from the classification

expected by M_{n-k} to the misclassification by M_n as regressed data, and data that have changed from the misclassification by M_{n-k} to the classification expected by M_n as improved data. The data sets are classified using the models M_{n-1} and $M_n(k=1)$.

Step 2: Impact calculation. NEURECOVER calculates five impacts to identify the weights to be corrected: the difference in weights, w_{diff}; the backward impact on the regressed data, $back_{reg}$; the forward impact on the regressed data, fwd_{reg}; the backward impact on the improved data, $back_{imp}$; and the forward impact on the improved data fwd_{imp}. The backward impact is given by the gradient of the loss function, and the forward impact is given by the output value of the forward propagation. A large difference in weights w_{diff} is assumed to be the cause of changes in the prediction results. NEURECOVER therefore obtains a weight table w_n and M_n and a weight table w_{n-k} from M_{n-k} and calculates $w_{diff} = w_n - w_{n-k}$. The back impact is calculated as follows: $back = \frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial o_j}\frac{\partial oj}{\partial w_{i,j}}$. For the localized weights $w_{i,j}$, which connect the j^{th} neuron of the previous layer and the i^{th} neuron of the target layer, and the output activation value of the o_j neuron of the target layer. back thus depends on the loss function L, the output of the neuron o, the weights w and the inputs. NEURECOVER obtains $back_{reg}$ as the back impact on the regressed data and $back_{imp}$ as the back impact on the improved data. The forward impact fwd is calculated as follows: $fwd = o_i - w_{i,j}$ from the activation value o_i of the i^{th} neuron of the previous layer and the weights $w_{i,j}$. It therefore depends on the weights w and the inputs. NEURECOVER obtains fwd_{reg} as a direct impact on the regressed data and fwd_{imp} as direct impact direct on the improved data.

Note that NEURECOVER locates the weights of all fully connected layers, not just the last layer of the DNN model. It calculates the backward and forward impact for each layer of the DNN model. In [40], for hypothesis testing in the initial phase, only fully connected layers are examined, but in the future, the authors plan to correct the weights of convolution layers as well.

Step 3: Fault Localization by Set Operation. In the last step, the weights are ordered for each of the five impacts: w_{diff}, $back_{reg}$, fwd_{reg} and $back_{imp}$. Then, the five corresponding sets W_{diff}, B_{reg}, F_{reg}, B_{imp} and F_{imp} of the first N_g weights are obtained. Finally, the weights specified by the set operation in the following equation are defined as the focus of the DNN model repair.

$$W_{localized} = (B_{reg} \cap F_{reg}) \cap W_{diff} \backslash (B_{imp} \cap F_{imp}) \qquad (5)$$

Through equation 5, NEURECOVER identifies a $W_{localized}$ set of weights that have a large difference and affect the regressed data, while excluding weights that affect the improved data. We consider that the weights that affect the regressed data and the improved data have large values for both the backward and forward impacts. Therefore, the weights that affect the regressed data are given by $B_{reg} \cap F_{reg}$, and the weights that affect the enhanced data are

given by $B_{imp} \cap F_{imp}$. The tuning operation therefore removes the regression from the data during the fault location and repair of the DNN model.

3.6.6. OTHER APPROACHES, TOOLS AND ARTICLES OF INTEREST

Detecting Deep Neural Network Defects with Data Flow Analysis

Repairing Deep Neural Networks: Fix Patterns and Challenges

A Comprehensive Study of Autonomous Vehicle Bugs

3.7. PROCESS 7: VALIDATION OF CORRECTIVES ACTIONS

3.7.1. INTRODUCTION: STATE OF THE ART

Corrective action shall be validated to ensure that the system is now fully operational. Depending of the corrective action, validation can differ.

In case corrective action was on non AI component, classical validation strategies can be adressed to return in service of the system. It could be visual inspection, self-test, or specific diagnostic to proves that all functions are available and works as intented.

In the second case, when corrective action was on AI component, the validation shall ensure two things :

- fault is now fixed
- no regression is introduced in the system

Non regression for AI system are a young field of interest and new technics will emerge in the next decades. At this day, the most intuitive ways, but also the most expensive and time consuming technics, consists to requalify the new version of the AI component with the same test sets used during its development and add new scenario to take account the new situation.

One more efficient way is to only take a subset of the qualification tests and to quantify the risk to miss a regression. If the risk is very unlikely (quantification to define), then the subset can be acceptable.

New technics consists to define group of tests which can be considered as independent. In other word, functional paths are grouped into independent path. A correction only affects some paths, then only test groups defined to these specific functional paths can be used to qualify the new version of the AI component.

3.7.2. IMPACT AND ISSUES OF THE AI

The validation of corrective maintenance actions for AI component is a new field of research and is a new challenge for the next decade. The main point to be checked and demonstrated is the non-regression status.

Corrective action on a faulty AI software (or brick) has to remove a faulty behavior, but at the same time, one has to be sure that it does not produce additional miss behavior on other use cases, which were not failing before. This a tricky issue which is not yet wholly covered by the state of the art but in which alternative solutions are proposed.

3.8. LOGISTIC SUPPORT ANALYSIS

3.8.1. INTRODUCTION: STATE OF THE ART

Logistic Support Analysis process is an approach to consider as a real product, the support system associated to a main system and which includes all logistics physical elements and strategies to help main system maintaining its performance and operational availability during its whole life cycle.

For this purpose, a systematic and comprehensive analysis has to be conducted on an iterative basis through all phases of the system/equipment life cycle to satisfy supportability (supportability includes all elements of ILS required to operate and maintain the system/equipment objectives). The level of detail of the analysis and the timing of task performance shall be tailored to each system/equipment and shall be responsive to program schedules and milestones.

The LSA tasks are divided into five general sections:

- **Section 100**, Program Planning and Control;
- **Section 200**, Mission and Support Systems Definition;
- **Section 300**, Preparation and Evaluation of Alternatives;
- **Section 400**, Determination of Logistic Support Resource Requirements;
- **Section 500**, Supportability assessment.

Task 100 deals with preparation of the support according to supportability objectives. Organization, plan, task, review, documents and goals are defined.
Task 200 consists in identifying and documenting the pertinent supportability factors related to the intended use of the new system/equipment. Factors to be considered include mobility requirements, deployment scenarios, mission frequency and duration, basing concepts, anticipated service life, interactions with other systems/end items, operational environment, and human capabilities and limitations. It also consists in Identifying existing and planned logistic support resources which have potential benefits for use on each system/equipment concept under consideration, providing supportability, cost, and readiness related information into mission hardware and software standardization efforts, identifying recommended mission hardware and software standardization approaches which have utility due to cost, readiness, or supportability considerations and participate in the system/equipment standardization effort and identifying any risks associated with each constraint established. Comparison of different technics, solutions, methods shall also be considered. This phase is also used to evaluate design opportunities or new technics for support.

Task 300 consists to identify functional requirement (those which need to be maintained) and how to support them (diagnostic, maintenance tasks, spare parts, logistic resources, human resource). Also it consists to identify hazards, including hazardous material, hazardous waste, and environmental pollutants associated with those functions identified. The results of the failure modes, effects, and criticality analysis (FMECA), or equivalent analysis, shall be

analyzed to identify and document corrective maintenance task requirements. The FMECA or equivalent, shall be documented on system/equipment hardware and software and to the indenture level consistent with the design progression and as specified by the requiring authority. Preventive maintenance task requirements shall be identified by conducting a reliability centered maintenance (RCM) analysis in accordance with the detailed guidelines provided by the requiring authority. The RCM analysis shall be based on the FMECA data and documented in the LSAR or equivalent format approved by the requiring authority. Operations, maintenance, and other support tasks shall be identified through analysis of the functional requirements of the new system/equipment taking into account mission analysis, and scenarios/conditions under which the new system/equipment will be operated. The analysis shall examine each system function allocated to personnel and determine what operator or support personnel tasks are involved in the performance of each system function. Alternatives tradeoffs can also be conduct to find alternative solutions.

Main objective of task 400 is to analyze required operations and maintenance tasks for the new system/equipment to:

- Identify logistics-support resource requirements for each task.
- Identify new or critical logistic support resource requirements.
- Identify transportability requirements.
- Identify support requirements which exceed established goals, thresholds, or constraints.
- Provide data to support participation in the development of design alternatives to reduce O&S costs, optimize logistic support resource requirements, or enhance readiness.
- Provide data for preparation of required ILS documents (technical manuals, training programs, manpower and personnel lists, etc).

It is also to assess the impact of introduction of the new system/equipment on existing systems, identify sources of manpower and personnel to meet the requirements of the new system/equipment, determine the impact of failure to obtain the necessary logistic support resources for the new system/equipment, and determine essential logistic support resource requirements for a combat environment. It analyses life cycle support requirements of the new system/equipment prior to closing of production lines to assure that adequate logistic support resources will be available during the system/equipment's remaining life.

Main objectives of task 500 is to formulating a test and evaluation strategy to assure that specified supportability and supportability related design requirements are achieved, or achievable, for input into system test and evaluation plans. The test and evaluation strategy formulated shall be based upon quantified and supportability requirements for the new system/equipment; the supportability, cost, and readiness drivers; and supportability issues with a high degree of risk associated with them. Tradeoffs shall be conducted between the planned test length and cost and the statistical risks incurred. Potential test program limitations in verifying supportability objectives based on previous test and evaluation experience and the

resulting effect on the accuracy of the supportability assessment shall be documented.

3.8.2. IMPACT AND ISSUES OF THE AI

For AI component, the LSA is challenging. In term of logistics, task to conduct maintenance actions, upgrade software version, validate 297307

spare parts compatibility, obsolescence management and their deployment and implementation to the entire fleet or serial vehicles.

For automatic shuttles and autonomous vehicle, several actors are involved during the support phase, fleet operators, vehicle manufacturers, AI technologies and components providers...

Logistic and responsibilities shall be shared and set mandatory between all these actors and be based on LSA standards to enable better exchange and increase the safety of the global product

4. CONCLUSION

When it comes to data monitoring, the maintainer will need to be concerned both with the data to be recorded in real time, and with the ability to adapt his monitoring devices to the situations he encounters. Indeed, given that the boundary between acceptable and unacceptable system behavior is difficult to determine in advance without human intervention (in the case of near-accidents), on-board data acquisition systems will certainly need to be upgraded to complete data that may prove incomplete at first instance. These developments will also present challenges, as the repeatability and replicability of situations and behaviours is not currently guaranteed, unlike for conventional critical systems. The rapid evolution of technologies, and possibly of versions of deployed systems, will bring its share of additional difficulties.

As previously stated, qualifying an event as unacceptable is one of the main challenges in maintaining an ARTS in operational condition. The impossibility of specifying the expected behavior of an AI brick (mainly machine learning, which is trained and unspecified), makes this qualification stage less straightforward than for a conventional system (whether the fault is a hardware failure, or a functional shortcoming). It is likely that the maintainer will need to be able to re-use some of the design results, in particular simulations, to be able to vary the system parameters within a range and scenario close to the observed event to be able to qualify the event as an unacceptable event.

Once the event has been qualified, the diagnosis must cover the entire system development cycle. In the case of a classic critical system, it is the isolation of the root case within the system that will eventually give rise to an impact analysis of the development cycle, with a view to improving processes. In the case of an ARTS, as it is not possible to isolate the fault within the AI component, it is the definition elements, from specifications and scenarios to the various training and simulation databases, that will have to be inspected almost systematically.

Implementing corrective actions on certain AI bricks is currently a challenge at the state of the art. Although methods have been proposed, they cannot systematically demonstrate that unacceptable behavior will be corrected without introducing new regressions. The validation of these corrective actions is in any case largely impacted by this limitation of current knowledge. The possible need to re-simulate for many hours the operation of the updated software brick must be taken into account, particularly for the operational availability of the ARTS as a whole.

Although current practices in critical systems engineering are sufficient to control the updating of a traditional critical system, it is in the evaluation of unacceptable events on the one hand, and the validation of corrective actions on the other, where the main difficulties for ARTS updating consist in.