2.3 Implementing requirements using machine learning (ML)

Practical guidance – cross-domain

Authors: Rob Ashmore (Dstl), and Dr Radu Calinescu and Dr Colin Paterson (Assuring Autonomy International Programme)

As with any engineering artefact, assurance can only be provided by understanding the complex, iterative process employed to produce and use ML components, i.e. the machine learning lifecycle as shown in Figure 1. The machine learning lifecycle consists of four stages. The first three stages—Data Management, Model Learning, and Model Verification—comprise the activities by which machine-learnt models are produced. Accordingly, we use the term machine learning workflow to refer to these stages taken together. The fourth stage, Model Deployment, comprises the activities concerned with the deployment of ML models within an operational system, alongside components obtained using traditional software and system engineering methods. Brief descriptions of each of the stages are provided below, with more detailed guidance on each step provide in the relevant section of the BoK.

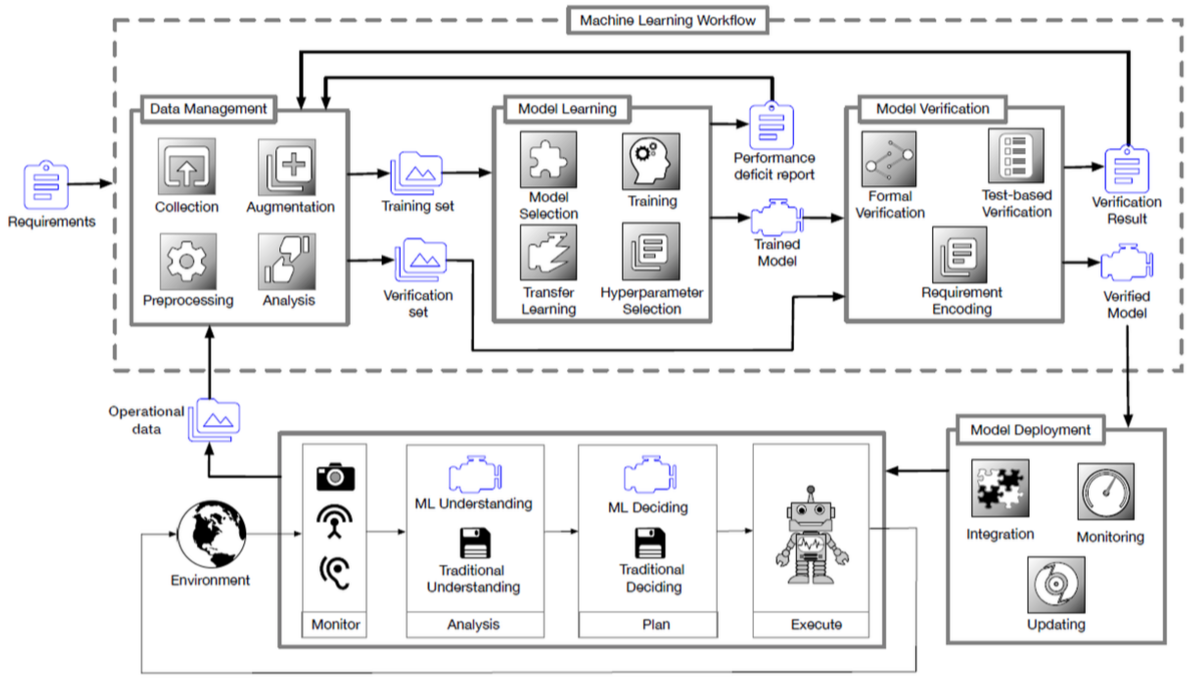


Figure 1 – The Machine Learning Lifecycle

Like traditional system development, the ML process is underpinned by a set of system-level requirements, from which the requirements and operating constraints for the ML models are derived. As an example, the requirements for a ML model for the classification of British road signs can be derived from the high-level requirements for a self-driving car intended to be used in the UK. However, unlike traditional development processes, the development of ML models involves the acquisition of data sets, and experimentation [1, 2], i.e. the manipulation of these data sets and the use of ML training techniques to produce models of the data that optimise error functions derived from requirements. This experimentation yields a processing pipeline capable of taking data as input and of producing ML models as output which, when integrated into the system and applied to data unseen during training, achieve their requirements in the deployed context.

Data Management

Data is at the core of any application of machine learning. As such, the ML lifecycle starts with a Data Management stage. This stage is responsible for the acquisition of the data underpinning the development of machine learnt models that can then be used "to predict future data, or to perform other kinds of decision making under uncertainty" [3]. This stage comprises four key activities, and produces the training data set and verification data set used for the training and verification of the ML models in later stages of the ML lifecycle, respectively. The first data management activity, collection [4, 5], is concerned with gathering data samples through observing and measuring the real-world (or a representation of the real-world) system, process or phenomenon for which an ML model needs to be built. When data samples are unavailable for certain scenarios, or their collection would be too costly, time consuming or dangerous, augmentation methods [6, 7] are used to add further data samples to the collected data sets. Additionally, the data may have limitations, and therefore preprocessing [8, 9] may be required to produce consistent data sets for training and verification purposes. Preprocessing may also seek to reduce the complexity of collected data or to engineer features to aid in training [10, 11]. Furthermore, preprocessing may be required to label the data samples when they are used in supervised ML tasks [4, 12, 3]. The need for additional data collection, augmentation and preprocessing is established through the analysis of the data [13].

More detailed guidance on data management is provided in section 2.3.1 of the BoK.

Model Learning

In the Model Learning stage of the machine learning lifecycle, the ML engineer typically starts by selecting the type of model to be produced. This model selection is undertaken with reference to the problem type (e.g. classification or regression), the volume and structure of the training data [14, 15], and often in light of personal experience. A loss function is then constructed as a measure of training error. The aim of the training activity is to produce an ML model that minimises this error. This requires the development of a suitable data use strategy, so as to determine how much of the training data set* should be held for model validation**, and whether all the other data samples should be used together for training or "minibatch methods" that use subsets of data samples over successive training cycles should be employed [12]. The ML engineer is also responsible for hyperparameter selection, i.e. for the choosing the parameters of the training algorithm. Hyperparameters control key ML model characteristics such as overfitting, underfitting and model complexity. Finally, when models or partial models that have proved successful within a related context are available, transfer learning enables their integration within the new model architecture or their use as a starting point for training [16, 17, 18]. When the resulting ML model achieves satisfactory levels of performance, the next stage of the ML workflow can commence. Otherwise, the process needs to return to the Data Management stage, where additional data are collected, augmented, preprocessed and analysed in order to improve the training further.

More detailed guidance on model learning is provided in section 2.3.2 of the BoK.

- * the training data set is the data that is produced independently of the verification data for use as part of the model learning process
- ** model validation represents the frequent evaluation of the ML model during training, and is carried out by the development team in order to calibrate the training algorithm. This differs essentially from what validation means in software engineering (i.e. an independent assessment performed to establish whether a system satisfies the needs of its intended users)

Model Verification

The third stage of the ML lifecycle is Model Verification. The central challenge of this stage is to ensure that the trained model performs well on new, previously unseen inputs (this is known as generalization) [4, 12, 3]. As such, the stage comprises activities that provide evidence of the model's ability to generalise to data not seen during the model learning stage. A test-based verification activity assesses the performance of the learnt model against the verification data set that the Data Management stage has produced independently from the training data set. This data set will have commonalities with the training data, but it may also include elements that have been deliberately chosen to demonstrate a verification aim, which it would be inappropriate to include in the training data. When the data samples from this set are presented to the model, a generalization error is computed [19, 20]. If this error violates performance criteria established by a requirement encoding activity, then the process needs to return to either the Data Management stage or the Model Learning stage of the ML lifecycle. Additionally, a formal verification activity [21] may be used to verify whether the model complies with a set of formal properties that encode key requirements for the ML component (e.g. requirements associated with model robustness). Formal verification allows for important properties to be rigorously established before the ML model is deemed suitable for integration into the safety-critical system. As for failed testing-based verification, further Data Management and/or Model Learning activities are necessary when these properties do not hold. The precise activities required from these earlier stages of the ML workflow are determined by the verification result, which summarises the outcome of all verification activities.

More detailed guidance on model verification is provided in section 2.3.3 of the BoK.

Model Deployment

Assuming that the verification result contains all the required assurance evidence, a system that uses the now verified model is assembled in the Model Deployment stage of the ML lifecycle. This stage comprises activities concerned with the integration of verified ML model(s) with system components developed and verified using traditional software and systems engineering methods, with the monitoring of its operation, and with its updating through offline maintenance or online learning. How the model is deployed within the system is a key consideration for an assurance argument.

Model Deployment Activities

1. **Integration** - This activity involves integrating the ML model into the wider system architecture. This requires linking system sensors, together with any necessary processing, to the model inputs. Likewise, model outputs need to be provided to the wider system so that they can be acted upon. A significant integration-related consideration is protecting the wider system against the effects of the occasional incorrect output from the ML model.
2. **Monitoring**- This activity is concerned with the different types of monitoring that are appropriate to the deployment of an ML-developed model within a safety-critical system. These can be organised into four categories:
    a. Monitoring the inputs provided to the model. This could, for example, involve checking whether inputs are within acceptable bounds before they are provided to the ML model.
    b. Monitoring the environment in which the system is used. This type of monitoring can be used, for example, to check that the observed environment matches any assumptions made during the ML workflow [22].
    c. Monitoring the internals of the model. This is useful, for example, to protect against the effects of single event upsets, where environmental effects result in a change of state within a micro-electronic device [23].
    d. Monitoring the output of the model. This replicates a traditional system safety approach in which a high-integrity monitor is used alongside a lower-integrity item.
3. **Updating** - Software is expected to change during a system's life. Similarly, deployed ML models are expected to be updated during their lifetime; some applications may involve weekly, or even nightly, updates. This activity relates to managing and implementing these updates. Conceptually it also includes, as a special case, updates that occur as part of online learning (e.g. within the implementation of an RL-based model). However, since they are intimately linked to the model, these considerations are best addressed within the Model Learning stage.

Desired Assurance Properties of a deployed ML model

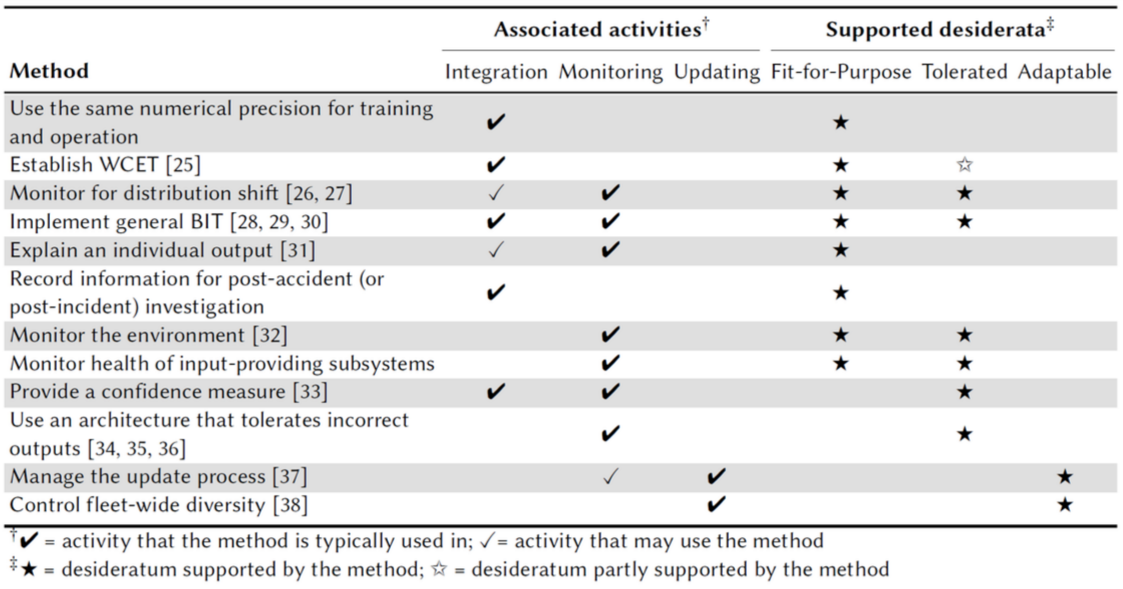
From an assurance perspective, the deployed ML model should exhibit the following key properties:

1. **Fit-for-Purpose** - This property recognises that the ML model needs to be fit for the intended purpose within the specific system context. The intended purpose is defined by the requirements for the ML model derived from requirements at the system-level. In particular, it is possible for exactly the same model to be fit-for-purpose within one system, but not fit-for-purpose within another. Essentially, this property adopts a model-centric focus.
2. **Tolerated** - This property acknowledges that it is typically unreasonable to expect ML models to achieve the same levels of reliability as traditional (hardware or software) components. Consequently, if ML models are to be used within safety-critical systems, the wider system must be able to tolerate the occasional incorrect output from the ML model.
3. **Adaptable** - This property is concerned with the ease with which changes can be made to the deployed ML model. As such, it recognises the inevitability of change

within a software system; consequently, it is closely linked to the 'Updating' activity described above.

Methods for Model Deployment

Table 1 provides a summary of the methods that can be applied during each Model Deployment activity in order to achieve the desired assurance properties (desiderata). Further details on the methods listed in Table 1 are available in [24].



	Associated activities[†]			Supported desiderata[‡]		
Method	Integration	Monitoring	Updating	Fit-for-Purpose	Tolerated	Adaptable
Use the same numerical precision for training and operation	✔			★		
Establish WCET [25]	✔			★	☆	
Monitor for distribution shift [26, 27]	✓	✔		★	★	
Implement general BIT [28, 29, 30]	✔	✔		★	★	
Explain an individual output [31]	✓	✔		★		
Record information for post-accident (or post-incident) investigation	✔			★		
Monitor the environment [32]		✔		★	★	
Monitor health of input-providing subsystems		✔		★	★	
Provide a confidence measure [33]	✔	✔			★	
Use an architecture that tolerates incorrect outputs [34, 35, 36]		✔			★	
Manage the update process [37]		✓	✔			★
Control fleet-wide diversity [38]			✔			★

[†]✔ = activity that the method is typically used in; ✓= activity that may use the method
[‡]★ = desideratum supported by the method; ☆ = desideratum partly supported by the method

Table 1 – Assurance methods for model deployment

Summary of Approach

Implementing an ML component for use in a RAS requires explicit consideration of the assurance of each of the following stages of the ML lifecycle:

1. Data Management – see further guidance in section 2.3.1
2. Model Learning - see further guidance in section 2.3.2
3. Model Verification - see further guidance in section 2.3.3
4. Model Deployment

References

- [1] Tom M. Mitchell. 1997. Machine Learning. McGraw-Hill.
- [2] Matei Zaharia, Andrew Chen, Aaron Davidson, et al. 2018. Accelerating the machine learning lifecycle with MLflow. Data Engineering (2018), 39.
- [3] Kevin P. Murphy. 2012. Machine Learning: A Probabilistic Perspective. The MIT Press.
- [4] Aurélien Géron. 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
- [5] Kiri Wagstaff. 2012. Machine learning that matters. (2012). arXiv:1206.4656
- [6] German Ros, Laura Sellart, Joanna Materzynska, et al. 2016. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In IEEE Conf. on computer vision and pattern recognition. 3234–3243.

- [7] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. 2016. Understanding data augmentation for classification: when to warp?. In Int. Conf. on digital image computing: techniques and applications. IEEE, 1–6.
- [8] SB Kotsiantis, Dimitris Kanellopoulos, and PE Pintelas. 2006. Data preprocessing for supervised leaning. Int. Journal of Computer Science 1, 2 (2006), 111–117.
- [9] Shichao Zhang, Chengqi Zhang, and Qiang Yang. 2003. Data preparation for data mining. Applied artificial intelligence 17, 5-6 (2003), 375–381.
- [10] Jeff Heaton. 2016. An empirical analysis of feature engineering for predictive modeling. In SoutheastCon. IEEE, 1–6.
- [11] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2018. Feature engineering for predictive modeling using reinforcement learning. In 32nd AAAI Conf. on Artificial Intelligence.
- [12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. Deep learning. Vol. 1. MIT Press.
- [13] R-Bloggers Data Analysis 2019. How to use data analysis for machine learning. Retrieved February 2019
- [14] Azure-Taxonomy 2019. How to choose algorithms for Azure Machine Learning Studio. Retrieved February 2019
- [15] Scikit-Taxonomy 2019. Scikit - Choosing the right estimator. Retrieved February 2019
- [16] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In IEEE Conf. on computer vision and pattern recognition. 1717–1724.
- [17] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In European Conf. on Machine Learning. Springer, 699–707.
- [18] Sanatan Sukhija, Narayanan C Krishnan, and Deepak Kumar. 2018. Supervised heterogeneous transfer learning using random forests. In ACM India Joint Int. Conf. on Data Science and Management of Data. ACM, 157–166.
- [19] Partha Niyogi and Federico Girosi. 1996. On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. Neural Computation 8, 4 (1996), 819–842.
- [20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15, 1 (2014), 1929–1958.
- [21] Luckcuck, M., Farrell, M., Dennis, L., Dixon, C. and Fisher, M., 2018. Formal specification and verification of autonomous robotic systems: A survey. arXiv preprint arXiv:1807.00048.
- [22] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. 2016. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. In V2CPS@IFM. 79–90.
- [23] A Taber and E Normand. 1993. Single event upset in avionics. IEEE Trans. on Nuclear Science 40, 2 (1993), 120–126.
- [24] Ashmore, R., Calinescu, R. and Paterson, C., 2019. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. arXiv preprint arXiv:1905.04223.

- [25] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, et al. 2008. The worst-case execution-time problem overview of methods and survey of tools. ACM Trans. on Embedded Computing Systems (TECS) 7, 3 (2008), 36.
- [26] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. Pattern Recognition 45, 1 (2012), 521–530.
- [27] Rob Ashmore and Matthew Hill. 2018. Boxing Clever: Practical Techniques for Gaining Insights into Training Data and Monitoring Distribution Shift. In Int. Conf. on Computer Safety, Reliability, and Security. Springer, 393–405.
- [28] Michael J Pont and Royan HL Ong. 2002. Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study. In First Nordic Conf. on Pattern Languages of Programs.
- [29] Muhammad Taimoor Khan, Dimitrios Serpanos, and Howard Shrobe. 2016. A rigorous and efficient run-time security monitor for real-time critical embedded system applications. In 3rd World Forum on Internet of Things. IEEE, 100–105.
- [30] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In Int. Conf. on Computer Safety, Reliability, and Security. Springer, 205–219.
- [31] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In 22nd ACM SIGKDD Int. Conf. on knowledge discovery and data mining. ACM, 1135–1144.
- [32] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. 2016. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. In V2CPS@IFM. 79–90.
- [33] Jasper van der Waa, Jurriaan van Diggelen, Mark A Neerincx, and Stephan Raaijmakers. 2018. ICM: An intuitive model independent and accurate certainty measure for machine learning.. In ICAART (2). 314–321.
- [34] Chris Bogdiukiewicz, Michael Butler, Thai Son Hoang, et al. 2017. Formal development of policing functions for intelligent systems. In 28th Int. Symp. on Software Reliability Engineering (ISSRE). IEEE, 194–204.
- [35] Paul Caseley. 2016. Claims and architectures to rationate on automatic and autonomous functions. In 11th Int. Conf. on System Safety and Cyber-Security. IET, 1–6.
- [36] Liming Chen, Algirdas Avizienis, et al. 1995. N-version programminc: A fault-tolerance approach to rellablllty of software operatlon. In 25th Int. Symp. on Fault-Tolerant Computing. IEEE, 113.
- [37] RTCA. 2011. Software Considerations in Airborne Systems and Equipment Certification. Technical Report DO-178C.
- [38] Rob Ashmore and Bhopinder Madahar. 2019. Rethinking Diversity in the Context of Autonomous Systems. In Engineering Safe Autonomy, 27th Safety-Critical Systems Symposium. 175–192.