MLGuard: Defend Your Machine Learning Model!

Sheng Wong
Scott Barnett
wongsh@deakin.edu.au
scott.barnett@deakin.edu.au
Deakin University
Melbourne, Victoria, Australia

Jessica Rivera-Villicana
jessica.rivera.villicana@rmit.edu.au
RMIT University
Melbourne, Victoria, Australia

Anj Simmons
Hala Abdelkader
a.simmons@deakin.edu.au
h.abdelkader@deakin.edu.au
Deakin University
Melbourne, Victoria, Australia

Jean-Guy Schneider
Jean-Guy.Schneider@monash.edu
Monash University
Clayton, Victoria, Australia

Rajesh Vasa
rajesh.vasa@deakin.edu.au
Deakin University
Melbourne, Victoria, Australia

ABSTRACT

Machine Learning (ML) is used in critical highly regulated and high-stakes fields such as finance, medicine, and transportation. The correctness of these ML applications is important for human safety and economic benefit. Progress has been made on improving ML testing and monitoring of ML. However, these approaches do not provide i) pre/post conditions to handle uncertainty, ii) defining corrective actions based on probabilistic outcomes, or iii) continual verification during system operation. In this paper, we propose MLGuard, a new approach to specify contracts for ML applications. Our approach consists of a) an ML contract specification defining pre/post conditions, invariants, and altering behaviours, b) generated validation models to determine the probability of contract violation, and c) an ML wrapper generator to enforce the contract and respond to violations. Our work is intended to provide the overarching framework required for building ML applications and monitoring their safety.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

design by contract, error handling, system validation, ML validation

ACM Reference Format:
Sheng Wong, Scott Barnett, Jessica Rivera-Villicana, Anj Simmons, Hala Abdelkader, Jean-Guy Schneider, and Rajesh Vasa. 2023. MLGuard: Defend Your Machine Learning Model!. In *Proceedings of the 1st International Workshop on Dependability and Trustworthiness of Safety-Critical Systems with Machine Learned Components (SE4SafeML '23), December 4, 2023, San Francisco, CA, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3617574.3617859

SE4SafeML '23, December 4, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0379-9/23/12.
https://doi.org/10.1145/3617574.3617859

1 INTRODUCTION

Robustness in business software where the domain is well understood is achieved through software testing, and adherence to best practices and processes. However, for Machine Learning (ML) systems this is insufficient. ML systems are dependent on data input streams that are non-stationary. As a result, ML is behaviour is underspecified [5] in the presence of subtle changes in the data (i.e. data shift [15]). Data schema validation alone is insufficient as detecting violations of these conditions, e.g., out of distribution data [10], can only be done probabilistically. We hypothesise that robustness can be incrementally realised in the context of ML through an interface specification (contract) that a) operates on point-estimates and distributions, b) encapsulates modelling assumptions, and c) models uncertainty as a first class citizen.

To achieve robustness for ML, research has focused on testing against noisy or malicious input data [1, 6, 12, 22]. However, the world is non-stationary and assumptions made offline are not guaranteed to hold during system operation. In addition, runtime validation is required to handle ML specific failure modes (i.e. miscalibration [8] and performance degradation under data shift [15]) independent to how the model responds to adversarial examples. Our research provides the constructs and plumbing for software engineers to leverage existing algorithms that detect these failure modes (due to the difficulty of detecting these failure modes, algorithms in turn may require additional ML models to perform the validation).

Studies have managed to produce data validation tools for ML systems that continuously monitor data that are fed into the system. Data Linter [11] acts as simplified and general validation tool that can be used to automatically inspect data and detect miscoding lints, outliers and scaling issues [11]. Data validation tools developed in-house like Deequ [17, 18] and Google's TFX allow user-specified constraints [2, 18]. Most of the validation tools available use constraints and conditions that are: deterministic in nature, provide partial support for ML failure modes, and typically act as a warning system for data with anomalous or invalid characteristics. In addition, data validation tools presented are insufficient in producing a reliable and robust ML system, since the "correctness" of data is variable depending on the context and scenario of the ML system.

Best practice recommends setting up alerting and monitoring infrastructure [2, 3, 18] and ML specific architectural tactics [4].

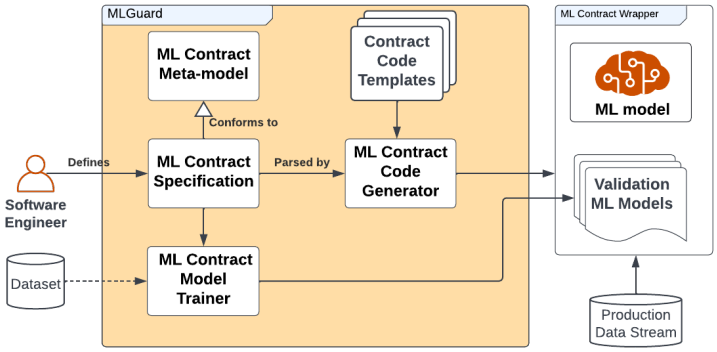

Figure 1: Our approach consisting of 1) an ML contract specification, 2) an ML contract model trainer to generate validation ML models that determine the probability of contract violations, and 3) a generated wrapper to defend models from contract violations and trigger contract violation handling logic if violations occur.

However, how to specify the conditions to monitor and the actions to take when a condition fails is left to the developer. ML toolsuites for validation (e.g., Tensorflow Data Validation) provide a set of validators and abstractions for their specification. However, these tools only offer partial support for ML failure modes and automation. Specifically, Tensorflow supports data schema and constraint suggestions, but leave more sophisticated validation checks such as out of distribution detection to the developer to implement and configure.

Inspired by concepts from design by contract [13], we propose a new approach, MLGuard, for specifying and validating ML contracts. Our approach takes an ML Contract Specification and generates an ML wrapper with *both the code and trained ML models for validating ML contracts*. We propose an ML contract specification language with i) ML specific concepts (e.g., 'uncertain') and ii) actions to take when the ML contract fails (e.g., log warning, throw exception, propagate uncertainty). To the best of our knowledge, this is the first proposed approach for specifying and validating probabilistic contracts for ML. Although no formal guarantees can be made of the absolute safety of the ML system, our approach provides a structured semi-automated way to help developers work towards improving the safety of the ML applications they develop by automatically detecting and responding to contract violations. MLGuard is designed to provide the scaffolding for specifying and validating contracts that, optionally, include additional validation algorithms i.e. a data drift detector.

2 MOTIVATION

To motivate our work, consider the case of software for automated epileptic seizure detection—classifying a segment of electroencephalogram (EEG) data as seizure or non-seizure. While a large number of papers have developed ML models for this task [20], these models come with (often undocumented) conditions that must be satisfied for the output produced by the model to be reliable. This poses a safety concern as over-interpretation of EEG data (false positives) might lead to incorrect diagnosis and treatment, and this causes numerous medication side-effects, driving restrictions, increased chance of mental illness and discrimination of job opportunities while under-diagnosis (false negatives) causes delayed treatment which might increase the risk of mortality and other physical injuries.

We elaborate on these challenges below, and propose a vision in section 3 to address them, thereby improving the safety of ML models when deployed in real-world applications.

No machine-checkable ML specification language. Well-designed software components document inputs, outputs, types, pre/post conditions to be satisfied, and exceptions that may be raised. However, ML models lack a full machine-checkable specification. For example, even though an ML model accepts a vector with the same type and dimensions as the EEG data, this does not necessarily mean that the model is a suitable choice. To determine if the model is compatible with the application, one also needs to consider whether the statistical characteristics of the training data and modelling assumptions match those of the application domain.

To assist in assessing the suitability of an ML model, proposals have been made for standardised documentation templates (i.e. data sheets [7] and model cards [14]). However, documentation templates require users of the ML model to manually read and interpret the ML model documentation (if any) without providing any machine-verifiable rules for safe use.

No mechanism to express uncertainty in validation rules. Different electrode placements, sampling frequencies, and filtering are possible. If these do not match those of the data the ML model was trained on, the ML model will still produce a result, but it cannot be trusted. The patient demographics may also affect the accuracy of the model. For example, it should not be assumed that a model trained on EEG from adults will work as well on EEG from children.

To ensure that the EEG data is compatible with the data on which the model was trained, one can make use of an out of distribution detection model to determine whether the assumptions of the model have been violated, i.e. the serving data in production should be distributed similarly to the data the model was trained on. However, this violation can only be detected probabilistically rather than with absolute certainty.

The conditions for these contracts operate in high-dimensional latent spaces. For ML, the input and learned latent spaces are where pre/post conditions are required to verify system behaviour. Currently we lack the mathematical constructs for guaranteeing behaviour across a high-dimensional latent space.

It is unclear how the application should respond to probabilistic violations. Unlike traditional software, ML behaviour is dependent on training data—demonstrating correctness offline is no guarantee of the system's operating behaviour online. Thus violations of contracts need to be detected and responded to during operation rather than at design time. Best practice recommends setting up existing alerting and monitoring infrastructure [3]. However, what is not specified is 1) how to configure alarms and alerts, and 2) how the system should respond.

3 A VISION FOR ML CONTRACTS

We propose MLGuard for automatically validating whether incoming data conforms to an ML contract and handling violations. MLGuard is a practical approach for dealing with the limitations outlined in the Motivation. An overview of our proposed approach is presented in Fig. 1, and more detailed descriptions of each MLGuard component are provided below.

3.1 ML Contract Meta-model and Specification

The ML Contract Meta-model provides the abstractions needed to specify ML contracts, serving as the basis for a **machine-checkable ML specification language**. For example, in addition to validating the data schema, one can specify that the model requires input data to be distributed similarly to the training data. The meta-model also provides the abstractions to define strategies for detecting probabilistic violations and for responding to probabilistic violations (the components to support this are elaborated on in the following sections).

We borrow the concept of declarative definition of constraints developed by Deequ [17, 18], and extend these ideas to allow for probabilistic conditions and specify actions to take. This approach will enable specifications be made regarding i) probabilistic conditions on inputs, ii) what methods will be used to detect probabilistic violations, and iii) approaches for dealing with probabilistic outcomes. For example, what should happen when conditions are violated with a confidence of 55% produced by a ML system. Software engineers write an ML Contract Specification tailored to their application needs that instantiates concepts in the meta-model. A sample contract is provided in Listing 1 using a YAML based syntax, but in future we will explore use of domain specific languages and fluent APIs.

3.2 ML Contract Model Trainer

Validating compliance with the ML Contract Specification requires Validation ML Models to detect probabilistic violations of contracts. The type of Validation ML Model to use may be specified as part of the ML Contract Specification, along with configurable thresholds at which to trigger contract violation handling logic, which together form a **mechanism to express uncertainty in validation rules**. For example, to validate that an instance of the input data is from the same distribution as the data the ML model was trained on,

```
Contract:
  Model:
    Name: seizure_detection_ml_model
    Location: /pretrained/seizure_model.onnx
    Documentation: /doc/seizure_model_card.html
  Data:
    - input_steam
    - output_stream
    - /data/eeg_train
  Preconditions:
    Distribution_Matches:
      DatasetA: input_steam
      DatasetB: /data/eeg_train
      Validation_model:
        Type: out_of_distribution_detector
        Method: likelihood_ratios_for_ood
      Trigger_conditions:
        Confidence_threshold: 0.95
      Action_if_violated: log_warning
    Schema_Matches:
      Dataset: input_steam
      Schema: /schema/eeg-10-20-system-256hz
      Action_if_violated: exception
  Postconditions:
    Probabilities_sum_to_one:
      Dataset: output_stream
      Action_if_violated: exception
```

Listing 1: Sample ML Contract Specification for seizure detection ML model. The ML contract allows specifying preconditions making use of probabilistic concepts (e.g., distributions match), methods for detecting probabilistic violations (e.g., out of distribution detection), and how to respond to probabilistic violations (e.g., whether to log a warning or raise an exception).

one may make use of an out of distribution detector. In the case of the Likelihood Ratios for Out-of-Distribution Detection method [16], this requires training deep-generative models to determine the probability that the data is out of distribution and correct for background statistics.

The role of the ML Contract Model Trainer is to automatically train the Validation ML Models (not to be confused with the ML Model that they are guarding) according to the configuration provided in the ML Contract Specification. To support software engineers uncertain about which type of ML Validation Model to use and how to configure it, we intend to explore approaches based on AutoML [9] to automatically select and train appropriate Validation ML Models to enforce the constraints in the ML Contract Specification when the Validation ML Model to use is left unspecified. Our approach will also allow for threshold conditions at which to trigger a violation to be automatically learned from data and refined based on user feedback in the case the the software engineer is uncertain about which threshold value to specify.

3.3 ML Contract Wrapper

A standard approach to addressing the issue of robustness is to introduce a wrapper [19] to guard against invalid behaviour. The ML Contract Code Generator selects and instantiates Contract Code Templates with information in the ML Contract Specification. The generated wrapper code includes the i) trained ML model, ii) Validation ML Models for use in pre/post conditions, and iii) code for checking pre/post conditions (using the Validation ML Models) to

guard the trained model and trigger contract violation handling logic to **respond to probabilistic violations**. The wrapper can be configured (via the ML Contract Specification) to respond to contract violations in a manner appropriate to the application and nature of violation. For example, should an exception be thrown, error messages logged, or uncertainty be propagated through the system?

4 RESEARCH QUESTIONS

The work proposed and discussed in the previous sections led us to pose the following research questions. Our plan to answering them is further discussed in section 5.

- **RQ1** What are the abstractions required for specifying ML contracts?
- **RQ2** What software architecture is required to enable the generation of a wrapper for enforcing ML contracts?
- **RQ3** How effective is an ML contract in practice?

5 FUTURE PLANS

Our research will progress in three phases. Phase 1 will focus on extracting concepts for the ML contract meta-model from the literature and defining the ML contract specification language. Phase 2 will involve experimental evaluation of the the ML Contract. Finally, Phase 3 will investigate the effectiveness of MLGuard in an industry context.

Phase 1: ML Contract meta-model and specification language: To answer research question *RQ1: What are the abstractions required for specifying ML contracts?* we will expand an existing specification language. We plan to follow an iterative approach inspired by Grounded theory [21] to mine concepts from the literature (both academic and grey literature). The goal of this phase is to identify the concepts for specifying ML Contracts and defining a validation plan. The expected outcome from this phase will be 1) the ML Contract meta-model, 2) an ML Contract specification language, and 3) a set of ML specific conditions for validation.

Phase 2: Experimental evaluation of ML Contracts: The next phase of research will involve developing a prototype of our solution to answer *RQ2: What software architecture is required to enable the generation of a wrapper for enforcing ML contracts?* Concepts borrowed from Model Driven Engineering (MDE) will be applied to design the generators (code and trained models). Our experiment will evaluate ML Contracts against other automated and manual approaches to specifying validation logic for ML. The expected outcomes from this phase will be 1) a prototype tool MLGuard, 2) code templates for ML Contract wrappers, 3) configurations for training Validation ML models, and 4) a set of ML Wrappers generated for existing models.

Phase 3: Industry case study: To address *RQ3: How effective is an ML contract in practice?* the final phase of the study will evaluate how our approach can be integrated into existing software engineering workflows. We plan to run a series of industry case studies where practitioners evaluate MLGuard on ML projects. The focus of the case studies will be to identify i) user-acceptance of the approach, ii) barriers to adoption, and iii) ongoing maintenance implications.

ACKNOWLEDGMENTS

The research was supported by a Deakin University Postgraduate Research Scholarship (DUPR) and a National Intelligence Postdoctoral Grant (NIPG-2021-006).

REFERENCES

[1] Leonhard Applis, Annibale Panichella, and Arie van Deursen. 2021. Assessing Robustness of ML-Based Program Analysis Tools using Metamorphic Program Transformations. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1377–1381.

[2] Denis Baylor et al. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1387–1395.

[3] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. 2017. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. In *Proceedings of IEEE Big Data*.

[4] Alex Cummaudo, Scott Barnett, Rajesh Vasa, John Grundy, and Mohamed Abdelrazek. 2020. Beware the Evolving 'Intelligent' Web Service! An Integration Architecture Tactic to Guard AI-First Components. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 269–280.

[5] Alexander D'Amour et al. 2022. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *J. Mach. Learn. Res.* 23, 1, Article 226 (2022).

[6] Xiang Gao, Ripon K Saha, Mukul R Prasad, and Abhik Roychoudhury. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1147–1158.

[7] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92.

[8] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International Conference on Machine Learning*. PMLR, 1321–1330.

[9] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.

[10] Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*.

[11] Nick Hynes, D Sculley, and Michael Terry. 2017. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. *NeurIPS* (2017).

[12] Marta Kwiatkowska. 2020. Safety and robustness for deep learning with provable guarantees. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 1–3.

[13] Bertrand Meyer. 1992. Applying "design by contract". *Computer* 25, 10 (1992), 40–51.

[14] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.

[15] 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.

[16] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. 2019. Likelihood ratios for out-of-distribution detection. *NeurIPS* (2019).

[17] Sebastian Schelter, Stefan Grafberger, Philipp Schmidt, Tammo Rukat, Mario Kiessling, Andrey Taptunov, Felix Biessmann, and Dustin Lange. 2019. Differential data quality verification on partitioned data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1940–1945.

[18] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1781–1794.

[19] Ali Shahrokni and Robert Feldt. 2013. A systematic review of software robustness. *Information and Software Technology* 55, 1 (2013), 1–17.

[20] Afshin Shoeibi et al. 2021. Epileptic Seizures Detection Using Deep Learning Techniques: A Review. *International Journal of Environmental Research and Public Health* 18, 11 (2021).

[21] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*. 120–131.

[22] Xiyue Zhang. 2020. Uncertainty-guided testing and robustness enhancement for deep learning systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 101–103.

Received 2023-07-04; accepted 2023-08-10