Truong, Linh; Nguyen, Tri

QoA4ML – A Framework for Supporting Contracts in Machine Learning Services

Published in:
2021 IEEE International Conference on Web Services (ICWS)

DOI:
10.1109/ICWS53863.2021.00066

Published: 15/11/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Truong, L., & Nguyen, T. (2021). QoA4ML – A Framework for Supporting Contracts in Machine Learning Services. In C. K. Chang, E. Damiani, J. Fan, P. Ghodous, M. Maximilien, Z. Wang, R. Ward, & J. Zhang (Eds.), *2021 IEEE International Conference on Web Services (ICWS)* (pp. 465-475). IEEE.
https://doi.org/10.1109/ICWS53863.2021.00066

QoA4ML – A Framework for Supporting Contracts in Machine Learning Services

Hong-Linh Truong, Tri-Minh Nguyen
Department of Computer Science, Aalto University, Finland
{*linh.truong,tri.m.nguyen*}*@aalto.fi*

***Abstract*—Important service-level constraints in machine learning (ML) services must be communicated and agreed among relevant stakeholders. Due to the lack of studies and support, it is unclear which and how ML-specific attributes and constraints should be specified and assured in service contracts for ML services. This paper examines service contracts in the three stakeholders engagement model of ML services. We identify key ML-specific attributes that should be specified and monitored for the ML service provider, ML consumer and ML infrastructure provider. Based on that, we propose QoA4ML (Quality of Analytics for ML) as a framework to support ML-specific service contracts. QoA4ML includes an ML-specific service contract specification, monitoring utilities and a contract observability service. To illustrate the usefulness of QoA4ML, we present real-world examples for contract terms and policies, monitoring and contract evaluation with dynamic ML services in predictive maintenance.**

1. Introduction

Machine Learning (ML) has been used pervasively in many application domains. Besides the development of ML models/algorithms, current research and industrial works are focused on developing ML tools and ML serving platforms [1], [2]. Such developments have fostered the adoption of a new business model – ML as a service (MLaaS) [3]: an ML provider can develop ML models and deploy these models as software services in a suitable ML infrastructure for customers. In MLaaS, there are different business engagement models, which fall into two main categories. First, in the *two stakeholders engagement model*, companies develop ML models and provision both ML services encapsulating these models and corresponding ML infrastructures for running ML services. Two main stakeholders exist: the ML service provider and the ML service customer. Examples of this case are ML services that Amazon, Google and Microsoft offer to many ML customers. Second, in *the three stakeholders engagement model*, an ML provider develops ML models and deploys the models in a third-party ML infrastructure for the provider's customers. In this case, the stakeholders are: the ML service provider, the ML service customer and the ML infrastructure provider; the ML service provider is a consumer of the ML infrastructure provider. We focus on MLaaS for the three stakeholders engagement model as it has a wider range of applications and businesses.

Although service contracts (also known as service level agreements – SLAs) have intensively been studied in cloud and enterprise computing, they are not well researched in the context of emerging MLaaS, which brings many new ML-specific attributes and constraints [4], [5], [6] due to the novel characteristics of ML and ML impacts on businesses. In particular, ML services for dynamic on-demand inference, such as making prediction with near real-time IoT data, are required to fulfill various runtime accuracy, responsive and bias constraints. For example, ML inference accuracy is a crucial attribute but it is dependent on various factors of ML models, input data and underlying computation. In the case of IoT data, the quality of data might vary a lot and the platform services and ML infrastructures have to cope with the speed and changes of input data. However, a violation of the inference accuracy might lead to strong (harmful) consequences in regulation compliance and business. Another contract aspect is that the ML consumer must be able to understand how ML services make decisions, at least through the trace of decisions within ML models [7], because the resulting inference might lead to unequal treatments, violating regulations. We see that contracts for ML services are part of the answer for the reliable and responsible ML [8], [9], helping to optimize the ML serving.

In this paper, we perform an analysis of ML attributes for interactions among stakeholders. We contribute the QoA4ML (Quality of Analytics for ML) framework for supporting services contracts in MLaaS. QoA4ML presents an abstract service contract specification for MLaaS that consists of ML-specific attributes such as performance, data quality, inference accuracy, privacy, fairness, and interpretability. The QoA4ML specification enables the developer and the provider to associate ML contract constraints with deployed ML services. To monitor ML attributes and constraints based on the QoA4ML specification, we develop monitoring utilities for capturing runtime attributes and an observability service for evaluating ML-specific contract values. We provide a prototype of the specification and implement validation policies using the Open Policy Agent framework [10] so that the QoA4ML framework can be easily integrated with state-of-the-art monitoring tools and ML systems in the edge and cloud environments.

The rest of this paper is organized as follows: Section 2 discusses the motivation of our work. Section 3 details QoA4ML's models and implementation. We present experiments in Section 4. We discuss related work in Section 5 and concludes the paper and outlines future work in Section 6.

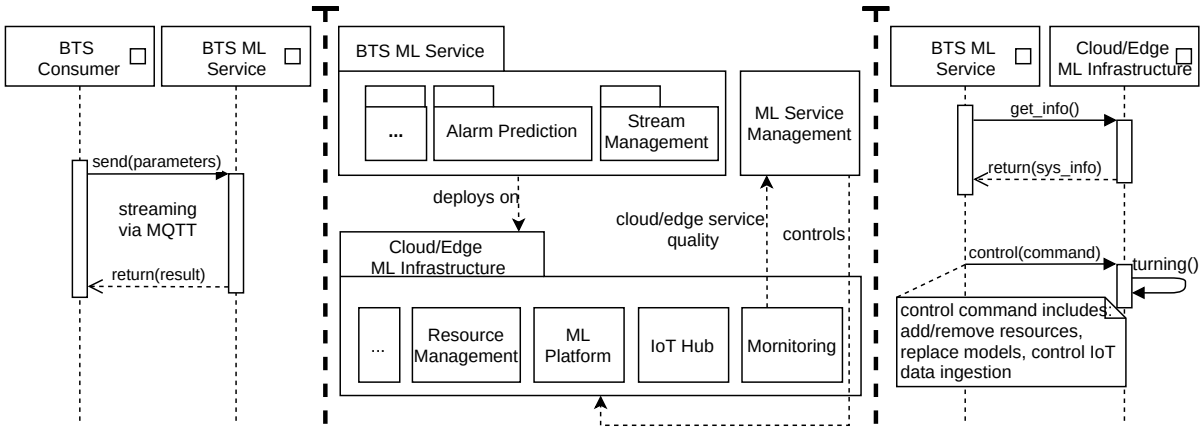

Figure 1. A simplified view of service interactions in BTS MLaaS. An ML service consists of and interacts with many components.

2. Motivation

2.1. Predictive Maintenance Scenario

A telco TC operates many base transceiver stations (BTS). In each BTS, many equipment and infrastructures need to be monitored for predictive maintenance purposes, e.g., the electricity from the grid supply, power generator, air conditioner, and power backup system. TC contracts a company $M(BTS)$ to perform all the monitoring and analysis of alarms in BTS. To enable predictive maintenance of the equipment, $M(BTS)$ relies on different ML services offered for a specific type of equipment. An ML service provider can offer an ML service – $S(E)$ for a type of equipment. In addition to that, $M(BTS)$ also offers its own service $S(BTS)$ for TC and other telcos. Here we have $M(BTS)$ as an ML consumer as well as an ML provider. (BTS) is a dynamic ML serving: serving inferences in near real-time for IoT-based data. Figure 1 shows service interactions. Since $M(BTS)$ can deploy its $S(BTS)$ on third-party ML infrastructures for TC and other telcos, $M(BTS)$ needs to clarify its ML-specific contracts. Currently, $S(BTS)$ is designed and deployed for both edge and cloud ML infrastructures.

In terms of service contracts, as an example, the inference accuracy is an ML-specific attribute whereas the response time is a common service attribute. The cost for ML services could be expensive, if higher inference accuracy and response time are changed elastically (e.g. due to the use of underlying models and fine-grained prediction). The ML service provider needs to agree on the tradeoffs of such attributes with the ML customer.

2.2. Research Questions

The above scenario highlights complex interactions among stakeholders in ML services. The ML service consumer, the ML service provider and the ML infrastructure provider need different types of agreements. The ML service provider might agree on common, non ML-specific aspects with the ML infrastructure provider. Between the ML provider and the ML service consumer, ML-specific aspects are crucial. We focus on the following research questions (RQs):

RQ1 – Which are the key attributes for ML contracts?: Besides various common QoS attributes and constraints found in cloud and enterprise computing, MLaaS brings new, important ML-specific attributes w.r.t technical runtime as well as legal and business aspects that must be supported in service contracts. To date, we have many papers studying ML attributes and their requirements and evaluation [5], [4], [11], [12]. In ML, key metrics such as inference accuracy and response time are often presented. However, corresponding ML contract attributes have not been defined and tested for ML services. MLaaS is a new model and so far it mainly follows the two stakeholders engagement.

RQ2 – How would ML attributes and constraints be specified?: ML-specific attributes and constraints must be specified in a way that they can coexist with other types of service constraints/configurations and be easily integrated into ML serving runtime systems as well as be used by different stakeholders. Furthermore, ML attributes/constraints must coexist well with other common complex service contracts in cloud services. Currently, the best practice of cloud/edge service uses various types of policies and languages written in JSON/YAML that can be easily deployed into the ML infrastructures. In the research, service contracts are usually based on formal models/languages. Therefore, they are not well positioned in the current support in MLaaS.

RQ3 – How would ML-specific attributes/constraints be monitored and evaluated?: It is hard to measure various ML-specific attributes and an end-to-end observability of these attributes is needed for contract monitoring. In case of ML services providing near real-time predictions, detecting and assuring the quality of input data within the services is challenging and the accuracy must be tested and reported on-demand. This requires an appropriate way that works well with suitable existing monitoring mechanisms in dynamic ML inference. For monitoring ML contracts, we need to

utilize suitable techniques for suitable attributes. Common attributes like response time and cost can be directly measured. However, to evaluate inference accuracy or quality of data attributes we may be required to use sampling, instrumentation and black-box testing techniques.

These questions are challenging because ML attributes have complex dependencies between models, data and computing through complex components, pipelines, and infrastructures. Our approach is to address ML-specific attributes and constraints first, before integrating them to other common, known aspects.

3. QoA4ML Framework

3.1. Important attributes for ML-specific contracts

Given an ML service – MLS, from the service contract perspective, we can consider a service contract $SC(MLS)$ as $SC(MLS) = (C_{SLA}, ML_{SLA})$ whereas:

- C_{SLA} representing well-studied service SLAs like cost due to resource consumption, service availability, security configuration, and resource elasticity.
- ML_{SLA} representing ML-specific SLAs like inference accuracy, fairness, membership inference privacy and quality of input data.

Our goal is to develop ML_{SLA} as this has not been well studied. ML_{SLA} replies on attributes associated with MLS in the serving phase (prediction/inference). Such attributes characterize the quality of analytics (QoA) of the ML. Many research papers and benchmarks have discussed key ML attributes [4], [12], [15] (also see the related work in Section 5). Therefore, in our work, we select important attributes for ML contracts. We propose to group ML-specific attributes for contracts into the following categories:

Inference Accuracy: reflects specific ML inference accuracy of the ML serving. Typically, MLS offers different ML models trained with a certain level of inference accuracy. During the serving, the accuracy of the resulting inference from these models might be varying. In some cases, service customers could evaluate, after a short delay, if the ML services give correct predictions (or with an acceptable/agreed level of accuracy). For instance in the BTS scenario, the maintenance company $M(BTS)$ could utilize a human-in-the-loop process to provide feedback on the accuracy of backup battery prediction through post-mortem batch analysis of the monitoring data and the resulting classifications. However, in other cases, it might not be possible to give the feedback due to the nature of the dynamic IoT anomaly prediction. The ML service provider needs to define constraints of accuracy and to utilize suitable methods to assure such constraints for the ML service consumer. Furthermore, the ML customer expects a mechanism to validate constraints and report problems w.r.t. inference accuracy. Generally, specific metrics for inference accuracy are dependent on providers. Our framework only provides a structure for capturing these metrics into contracts.

Reliability and Elasticity: refers to the capability of providing elasticity and reliability of ML inferences. Here we note that the elasticity and reliability are from the service function viewpoint: it is whether the ML services can offer elastic quality and inference features, e.g., to deal with different tradeoffs of performance loss and inference accuracy or reliable inferences under varying quality and distribution of input data. This aspect of reliability and elasticity is ML-specific and is in the responsibility of the ML service provider. It is not similar to the common service reliability and elasticity in C_{SLA} in ML infrastructures (e.g., service availability, service failure or resource elasticity), which is under the responsibility of the ML infrastructure provider. In this aspect, the ML inferences have to deal with changes of dynamic data volume and velocity. This might require the ML services to utilize different techniques to invoke parallel/concurrent inferences as well as to switch/select suitable ML models. Furthermore, since dynamic inferences rely on other service platform capabilities, such as message brokers, this ML-specific aspect is also strongly related to the elasticity and reliability of the ML infrastructures (from the infrastructure provider).

Quality of Data: includes quality of data attributes that strongly influence the ML inference. The ML service provider expects the input data with certain quality constraints to ensure the quality of the ML serving. Since the ML consumer provides the input data and the ML service provider performs inferences based on the quality of data, the ML consumer must assure quality of input data met constraints and the ML provider can reject serving or hold no assurance, if the quality of input data is not met. Another contract aspect is the ability to detect data drift/quality for the customer. This feature is important as in many situations the input data is collected and sent for inferences in real-time, e.g., from IoT devices.

Security and Privacy: refers to the ML service's security and private setting, in terms of stakeholder contracts, not internal security of the ML service, such as adversarial protection, data poisoning, and transfer learning protection [16], [17]. Common service security and privacy would be about secured communication, GDPR-compliant personal data handling, or restricted location of services. ML-specific attributes can be privacy risk measured from inference attack tests and inference support for encrypted input data.

Fairness and Interpretability: refers to the ability of ML models and results to be verified and interpreted. Fairness is the expectation from the ML consumer when using ML services. Service consumers expect to obtain and test measures of fairness to avoid unfair treatment and misuse of results. We thus mainly support (i) the customer to check the algorithmic fairness [11] caused by the service provider and (ii) the provider to check root-cause due to data bias. In principle, there might not be an absolute fairness guarantee in terms of service contracts. However, the ML service could provide additional information and fairness implementation in ML models. Interpretability details could be a "tracing" feature that the ML service provider can turn on/off on-demand [6]. Implementation of fairness and interpretability

Category	Examples of ML-specific attributes	Examples of contract constraints and possible monitoring
Inference Accuracy	Accuracy of the inference result, ROC, AUC, Root Mean Squared Error [5]	The ML service guarantees minimum 90% & 95% inference accuracy in an edge deployment and in a cloud deployment, respectively.
Reliability & Elasticity	Reliable output under concept drift, elastic accuracy via an ensemble of models	Detecting and dealing with concept drift must be supported; elasticity w.r.t. dynamic versus static inference.
Quality of data	Data accuracy & completeness, outlier of input data, data drift, data distribution	The ML service must specify outlier score and data drift report for the ML customer since data distribution change, ML models become outdated
Security and Privacy	Privacy risk from membership inference [13]	The ML consumer needs the probability of privacy risk due to membership inference less than 1%
Fairness	Demographic parity, equalized odds, equal opportunity, bounded group loss [11]	The ML service must not have the disparity in prediction performance higher than 15% (e.g., given gender as a representative sensitive data) and the ML consumer can periodically check using known data and result
Interpretability	Traces of ML inferences and explanations for inference results [7], [6], [14]	The ML consumer needs to obtain detailed trace for explaining the inference result (explainability mode)
Cost	The whole cost, breakdowns of cost for serving and for other ML tasks	The ML service must provide details costing items for inference versus for preparation and other activities

TABLE 1. EXAMPLES OF ML-SPECIFIC ATTRIBUTES AND CONSTRAINTS

features will strongly affect the performance, inference accuracy and cost. However, they help to address the liability of the ML service and of the ML consumer, if wrong things happen. Therefore, they must be part of the service contract. *Cost*: refers to the aspect of financial costs related to the ML inference results, instead of common resource and service costs in C_{SLA}. Between the ML service consumer and the ML service provider, cost elements might be dependent on, e.g., the number of requests, the accuracy of the result and the serving time. Between the ML service provider and the ML infrastructure provider, cost elements fall into common factors in C_{SLA}, such as infrastructural resource consumption and service platforms usage.

Table 1 shows examples of ML contract attributes and constraints. In principle, the list of attributes, which should be supported in ML service contracts, can be extended. From these categories, we define ML_{SLA} as a contractual means that consists of ML-specific attributes, establishing ML-specific (sub)contract in a general MLaaS service contract.

3.2. QoA4ML Specification

To support ML_{SLA} we introduce the abstract QoA4ML (Quality of Analytics for ML) specification. The abstract QoA4ML specification includes terms and constraints, which can be used to (i) implement ML-specific service contracts, (ii) monitor ML services and their contracts, and (iii) allow runtime changes and updates of ML services according to specific contexts. Figure 2 shows the structure of the abstract QoA4ML specification. The main categories of resources are *services*, *data* and *ML models*, whereas *services* can be further divided into different types, such as, for computing, storage and communication. The abstract QoA4ML specification contains term definitions and guidelines to build service contracts. Thus, a concrete implementation of the abstract QoA4ML can be based on a suitable syntax; we do not enforce the syntax in order to allow the flexible integration with underlying ML frameworks and common service contracts.

The QoA4ML specification covers two aspects:

- specifying required attributes and their constraints: this allows us to specify required attributes without indicating how we can verify them. Such a specification can be reused and allows us to use different techniques to verify the attributes and constraints.
- specifying policies as functions for evaluating attributes and their constraints: policies are functions used to evaluate the specified attributes and concrete observed behaviors to validate the contract. This allows us to use different languages to implement the policies as well as to integrate with different monitoring systems.

In principle, two aspects can be merged into a single language and toolset. However, separating them will help to improve the reusability, integration and adaptation of contracts at runtime. There are different ways to implement QoA4ML, such as formal models or high-level languages with corresponding tools. We will detail our implementation in the following.

3.2.1. QoA4ML Specification of Attributes and Constraints. Our current work is focused on the implementation that can be easily integrated with other runtime requirements in state-of-the-art ML services. Consider that ML services are strongly dependent on cloud and edge platform services whereas many other types of policies will have to be developed, our first implementation is to be based on JSON and the OPA framework for contract constraint validation [10].

The QoA4ML specification includes key terms that an ML-specific contract can reuse. Listing 1 presents an example of terms. The structure of an QoA4ML-based contract can follow a similar structure, given in Figure 2, using terms defined in Listing 1. For example, consider the BTS scenario, based on the QoA4ML specification, metadata about ML-specific attributes for the ML service – $S(BTS)$ – a contract can be defined in JSON as shown in Listing 2. Listing 2 shows an excerpt of the contract. `mlmodels` will be deployed on ML infrastructures within 2 types of computing resources: `small` (edge machine) and `normal` (cloud server). The contract defines maximum 0.1 seconds for response time, minimum 80% accuracy for input data, and minimum 80% ML inference accuracy.

Listing 2. Excerpt of a BTS contract for dynamic inference in the edge

```
"resources": {
```

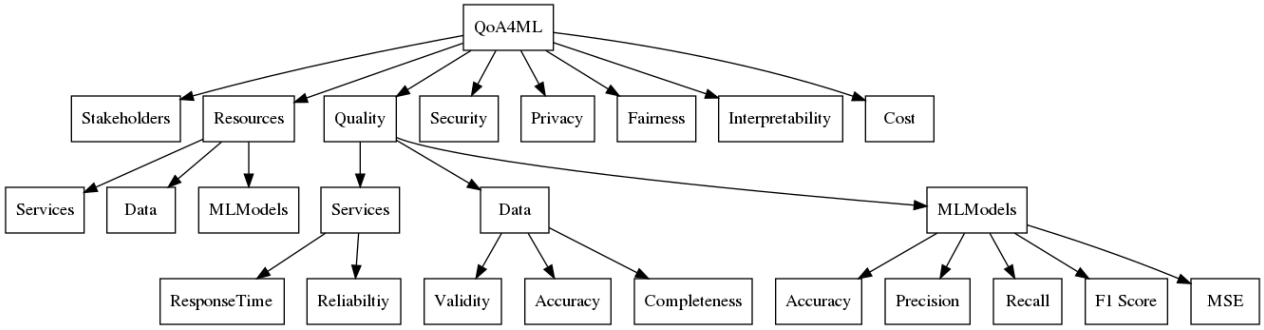

Figure 2. Examples of the structural elements in the QoA4ML specification

```
    "mlmodels":[
      {"id": "ml_inference", "mlinfrastructures": "
           tensorflow", "machinetypes": ["edge", "cloud"],
            "inferencemodes": "dynamic"}
    ]
  },
 "quality": {
    "services": [
            {"ResponseTime":{"operators":"max","unit":"s
                 ","value":0.05,"class": ["performance"],
                 "machinetypes": ["edge"]}}
    ],
    "data":[
            {"Accuracy":{"operators":"min","unit":"
                 percentage","value":80, "class": ["
                 qualityofdata"]}}
    ],
    "mlmodels": [
            {"Accuracy":{"operators":"min","unit":"
                 percentage","value":80, "class": ["
                 Accuracy"], "machinetypes": ["edge"]}}
    ]
 }
```

3.2.2. Policies for validating runtime QoA4ML attributes. The ML-specific attributes in our focus are observable: these attributes can be measured/estimated through observability techniques. While certain attributes can be reported and extracted inside ML services (can only be done by the ML provider/developer through instrumentation), others have to be observed outside the service by the ML provider or consumer (see Section 3.3). Based on the contract, policies for evaluating contract constraints can be generated/written. Currently, we implement the policies based on the Rego language in the OPA, e.g., shown in Listing 3 for the BTS contract. We use Rego/OPA because the ML-specific contract policies can also be easily integrated with other system/infrastructure policies. Both concrete contract and policy specifications are stored in the ML observability service. There are many steps that can be automated and required supporting tools, such as service discovery and user interface for contract definition and policy generation. However, we have not performed such works.

Listing 3. Example of policies for validating contract constraints

```
package qoa4ml.bts.alarm
import data.bts.contract as contract
default contract_violation = false
contract_violation = true {
    count(violation) > 0
}
# The policy checker will receive the contract and the
     runtime information
# input variable: the input of runtime metrics
violation[[input.client_info, "Accuracy violation on edge
      resource"]]{
    input.service_info.machinetypes == "edge"
    input.service_info.metric[_] == "Accuracy"
    some i, j
    contract.quality.mlmodels[i].Accuracy.machinetypes[j]
         == "edge"
    input.metric.Accuracy < contract.quality.mlmodels[i].
        Accuracy.value
}
violation[[input.client_info, "ResponseTime violation on
     edge resource"]]{
    input.service_info.machinetypes == "edge"
    input.service_info.metric[_] == "ResponseTime"
    some i, j
    contract.quality.services[i].ResponseTime.
        machinetypes[j] == "edge"
    input.metric.ResponseTime > contract.quality.services
        [i].ResponseTime.value
}
violation[[input.client_info, "Data quality violation"]]{
    input.service_info.metric[_] == "DataAccuracy"
    some i
    contract.quality.data[i].Accuracy.operators == "min"
    input.metric.DataAccuracy < contract.quality.data[i].
        Accuracy.value
}
```

3.3. Monitoring Utilities and Observability Service

Figure 3 shows our observability system. We develop different *QoA4MLprobes* as monitoring utilities for capturing ML-specific runtime attributes that will be instrumented into ML services and ML infrastructures as well as used by the ML consumer applications to produce QoA4ML reports. These probes, together with common monitoring features (e.g., for common contract metrics), collect different runtime attributes for evaluating QoA4ML. The probes are designed to work with common cloud and edge monitoring systems. In our case, the *Monitoring Service* is based on Prometheus [18]. QoA4ML contracts and policies are managed by the *QoA4ML Observability Service*, which is a microservice leveraging the OPA framework for evaluating contracts. In order to evaluate contracts at runtime, QoA4ML reports will be aggregated and sent to the QoA4ML Observability Service, which will check the policies and report any violation. Hence, the ML consumer or the ML service can build such `qoa4ml report` itself together with the Monitor-

Listing 1. Excerpt of core terms (simplified) in JSON in the QoA4ML specification

```
"resources": {
    "services":[
        {"id": "service_id", "serviceapis": ["rest", "mqtt","kafka", "amqp", "coapp"], "machinetypes":["micro","
            small","normal"],"processortypes":["CPU","GPU", "TPU"]}
    ],
    "data": [
        {"id": "data_id", "datatypes": ["video","image","message"], "formats": ["binary","csv","json","avro"]}
    ],
    "mlmodels": [
        {"id": "mlmodel_id", "formats": ["kerash5","onnx"], "mlinfrastructures": ["tensorflow","predictio"], "
            modelclasses":["SVM","DT","CNN","RNN","LR","KMeans","ANN"], "inferencemodes":["static","dynamic"]}
    ]
},
"quality": {
    "__comment": "when define a contract, we replace @ATTRIBUTE with a real attribute name",
    "services": {
        "@ATTRIBUTE":{   "id":"id",
            "attributenames":["ResponseTime","Reliability"], "operators":["min","max","range"],
            "unit":["ms","second","percentage"], "value":"value", "class":["performance"]}
    },
    "data": {
        "@ATTRIBUTE":{
            "id":"id",
            "attributenames":["Accuracy","Completeness"],"operators":["min","max","range"],
            "unit":["percentage"], "value":"value","class":["qualityofdata"]
        }
    },
    "mlmodels": {
        "@ATTRIBUTE":{
            "id":"id",
            "attributenames":["Accuracy","Precision","Recall","AUC","MSE"], "operators":["min","max","range"],
            "unit":["percentage"], "value":"value", "class":["accuracy"]
        }
    }
},
"security": {
    "encryption":{"types": ["end2end"]},
    "encryptedinference": {"mode":[true,false]}
},
"privacy": {
    "membershipinferencerisk":{"operators":["min","max"],"unit":"float","range":[0,1],"value":"value", "class": ["
        Privacy"]}
},
"fairness":{
    "predictionbias":{"operators":["min","max"],"unit":"percentage","value":"value", "class": ["Accuracy"]}
},
"interpretability": {
    "explainability": {"modes": ["full","compact"]}
},
```

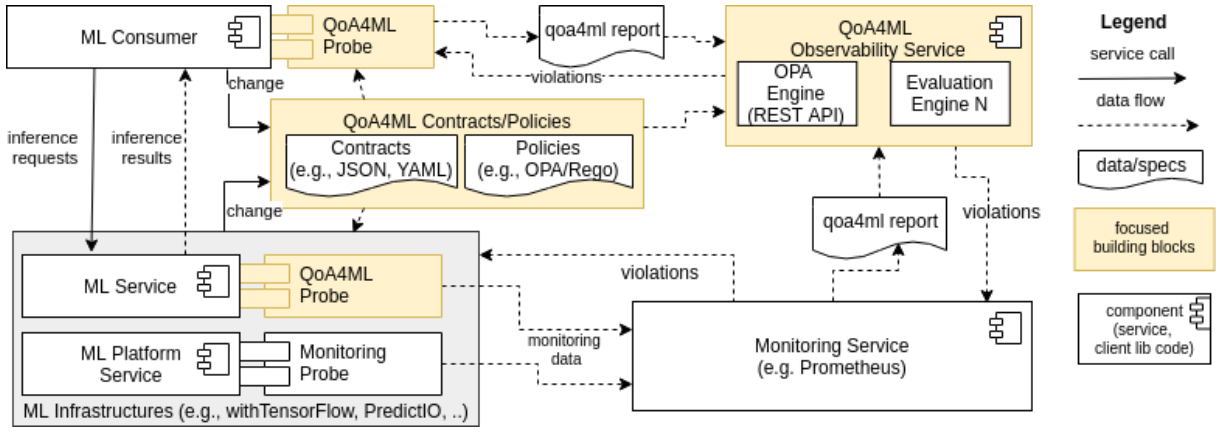

Figure 3. High-level view of probes, monitoring service and observability service and their interactions for ML service contracts

ing Service. For the evaluation of violation, we implement specific evaluation functions for OPA-based. An OPA-based engine service is deployed whereas contracts (ML_{SLA}) are described in JSON documents and rules for evaluation are described/written in Policies (Rego).

Monitoring utilities – QoA4ML probes – implemented as plugins can be used in different scenarios:

Integrating with the ML consumer application, service and

infrastructure: such probes should be integrated by the developer and the provider. The ML service developer exports measurements and metrics needed for ML-specific attributes using specific probes for making QoA reports. For example, our current Python-based probes can be instrumented into the ML service with a few lines of code. The ML service developers can measure metrics themselves e.g., using data quality tools to capture data accuracy and logging the ML inference accuracy and response time, then trigger our probes during serving requests for submitting those metrics to Observability Service. On the other hand, several probes are complex, capturing ML-specific attributes for the developer and they can be used for sampling infrastructure metrics or integrated into the existing monitoring system. Such probes are expected independent from ML services/consumer applications, e.g., using API wrappers (e.g., using API gateway and service mesh technologies) and external component invocations (e.g., sampling quality of data in a queue). *Integrating with human-in-the-loop feedback for ML:* such probes could be implemented as separated functions that are part of human-in-the-loop feedback processes for ML. Such processes can act as a stand-alone or be integrated into ML customer applications and monitoring tools. They allow the end-user to check and test the ML-specific attributes. Examples are (i) monitoring availability and response time, (ii) evaluating inference accuracy and fairness.

Note that certain probes, especially for measuring quality of data and fairness, are data-dependent. Therefore, such probes are customized for certain domains and types of data. In our current implementation for BTS, probes for data quality, inference accuracy and response time have been implemented. Other probes are currently being developed.

3.4. On dependencies among contract attributes

Many ML-specific attributes are related to common well-studied attributes, especially the case of the reliability, response time, and accuracy attributes. In this paper, we do not focus on identifying/defining their complex relationships. From the perspective of service contracts, the specification of attributes and constraints are independent from the study of such complex relationships. In our view, the stakeholders must understand such relationships to define constraints of ML attributes. Furthermore, the policies can be complex when there are many attributes. However, how to write the best policies for specific ML services is not our goal.

We believe that understanding the complexity of dependencies should be the subject of other papers. In particular, such dependencies might be very specific to ML services and domains. Hence the service stakeholders, especial the ML provider, must have a way to characterize the dependencies and reflect the work into attributes and constraints. If the dependencies are not studied carefully, either the service provider cannot provide complex attributes for the contract or cannot optimize the ML runtime. Both aspects are out of the scope of our work but they are important for the adoption of service contracts.

4. Illustrating Examples and Experiments

4.1. Experiment Settings

4.1.1. Experimental models. We illustrate monitoring of ML contracts with the BTS predictive maintenance scenario. While monitoring data is collected across many BTS, an ML service predicts anomaly behavior by using historical data captured from sensors in near real-time. In our experiments, the prediction is customized for each station regarding its conditions and settings. The prediction employs an attention-based neural network [19] using the core LSTM [20] (called LSTM model) to predict the *Load of Power Grid* at a single station. We trained the LSTM model using a real dataset provided by a company in Vietnam[1]. First, the BTS ML model is trained in the cloud and then exported for the edge, creating a version for the edge ML service, called `BTS-model-edge`. From the initial cloud version, we retrained it in the cloud several times, creating some other versions called `BTS-model-cloud`, which improves inference accuracy at different periods of a day. Since the goal of our experiments is not to study the prediction model, but possible ML contracts, we trained these versions of the LSTM model with a limited amount of data. All versions of the trained model are able to run on different ML infrastructures – either in the cloud or in the edge infrastructures using TensorFlow and TensorFlow Lite with Python. `BTS-model-edge` and `BTS-model-cloud` are loaded into the BTS ML service instances in the edge and cloud ML infrastructures, respectively.

4.1.2. Experimental ML serving platform deployment. Figure 4 shows the main configuration of our experiments. The customer application, illustrated by the `Python Application` was deployed in the edge that pushes IoT data for the ML service; it is assumed to run within a BTS and uses a message broker to communicate with the BTS ML service. When the BTS ML service is configured in the edge infrastructure, both the BTS ML service and the ML consumer application are deployed at the edge using Raspberry Pis. When the BTS ML service is configured in the cloud infrastructure, it is deployed within a container. Note that the edge deployment is not movable: our scenario is not a mobile edge cloud deployment.

The QoA4ML Observabilility Service and the Monitoring Service (Prometheus) are run on Pouta - CSC Cloud Service[2]. All the ML requests/responses and QoA4ML reports are transmitted via a messaging system, a common method for transferring data in IoT. In the dynamic ML serving context, we evaluated different deployments of message brokers on either the edge or the cloud for different cases.

4.1.3. Experimental contracts. For all experiments, we defined the contracts to examine three attributes: *Respon-*

1. https://github.com/rdsea/IoTCloudSamples/tree/master/MLUnits/BTSPrediction
2. https://research.csc.fi/-/cpouta

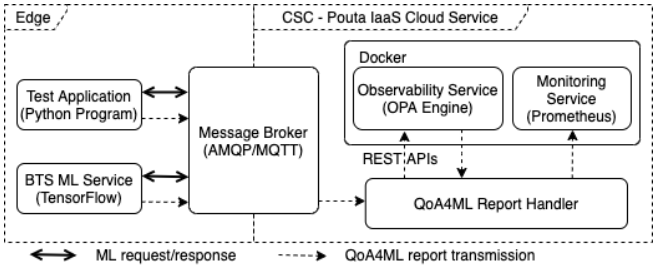

Figure 4. Experiment configurations for the BTS ML scenario

seTime, *Inference Accuracy*, and *Data Quality* (for the input data), as shown in Listing 2. Assuming the role of the ML provider, we can specify different conditions on these attributes that must be guaranteed in ML contracts between the ML service provider (running the BTS ML service) and the customer (the company doing predictive maintenance). For example, when serving the BTS ML model in the edge (`BTS-edge-model`), we assumed the `BTS-edge-model` could not be trained periodically due to limited resources, the expected *Inference Accuracy* could be 80%. When the BTS ML service was deployed in the cloud, with `BTS-model-cloud`, we expect the BTS ML service to achieve 90% accuracy or even higher. In terms of *ResponseTime*, clearly the underlying ML infrastructures and other platform services (e.g., message brokers) will have a strong impact on the response time, due to the network and compute resource configurations. *Data Quality* is considered in our contract as low-quality input data can affect the prediction, leading to potential disputes in the service contract. To experiment our framework, we changed the policies to demonstrate their deciding role in detecting violation (see Listing 3). For example, when *Inference Accuracy* does not meet the expectation because *Data Quality* is not satisfied, the system only counts a violation of *Data Quality*.

4.2. Experiments

4.2.1. Effect of edge and cloud serving platform deployment in ML contracts. In the first experiment, we used two deployments of message brokers (messaging systems for transferring IoT data and other types of observability reports): MQTT (HiveMQ Community Edition[3]) and AMQP (RabbitMQ[4]). The brokers were deployed in the edge and in the cloud to demonstrate the flexibility of changing contract constraints and violation detection of our framework with different system setups, which should be considered in the ML contract. Table 2 shows the violation rates observed when the customer application sent the prediction request based on the data of a day including more than 3.000 records within 15 minutes. Here, the *ResponseTime* in our ML contract is simply set as the average *ResponseTime* observed in each test case. When the brokers are deployed in the edge, all the communications are performed within the internal edge network. We can only recognize several violations due to unstable connections of the local network (home-setting wireless with Raspberry Pis due to COVID-19). With the brokers deployed in the cloud, AMQP shows only a few spikes causing violations, and most requests are responded on time (showed in Figure 5). Using MQTT broker, the *ResponseTime* fluctuates with bigger amplitudes (∼*70ms*) around the average value. Nearly a half of responses cause violations. We performed an additional experiment where we keep the broker in the edge and moved the ML service between edge and cloud. We simulated the edge environment using Google Cloud Platform. Table 3 shows that the violation rates when using external network are always higher. Since the internal network on Google service is more stable, we observed only a few spikes. The amplitude of the variation observed in *ResponseTime* is also smaller (∼*25ms*) so that if we just simply set the expected *ResponseTime* as the average value, there will be a lot of violations.

With dynamic configurations of ML services and their ML infrastructures for customers, the ML providers have to carefully examine the *ResponseTime* to make a suitable contract offering a flexible service to their customers. However, it will be difficult if they cannot find a suitable solution to dynamically monitor, change, update and deploy the ML service contract. QoA4ML allows developers to define ML attributes in the contract considering different system conditions and settings. Meanwhile, the violation monitoring would show the actual quality of service that ML providers deliver to their customers. Based on that, they can add, remove, or adjust the contract/policy flexibly in runtime due to condition changes. That builds a life cycle for the ML contract helping all the involved stakeholders avoid unexpected compensation.

Broker Deployment	Broker Type	Violation Rate
Edge (Raspberry PI, local network)	MQTT	12%
	AMQP	8%
CSC - Cloud	MQTT	41%
	AMQP	16%

TABLE 2. *ResponseTime* VIOLATIONS IN BROKER DEPLOYMENTS

ML Service Deployment	Broker Type	Violation Rate
Edge (container, Google Cloud)	MQTT	20%
	AMQP	18%
CSC - Cloud	MQTT	38%
	AMQP	21%

TABLE 3. *ResponseTime* VIOLATIONS WHEN MOVING ML SERVICE

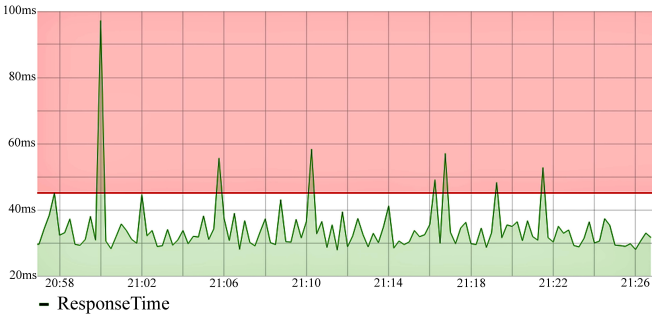

Figure 5. *ResponseTime* when the broker and ML service in the edge

3. https://www.hivemq.com/developers/community/
4. https://www.rabbitmq.com/

4.2.2. Impact of violation monitoring. In the second experiment, to demonstrate the importance of service contracts for *Inference Accuracy* and *Data Quality* attributes, our BTS ML model was trained with only some parts of the dataset as training the model with the whole dataset makes it hard to demonstrate these attributes with IoT data. In this experiment, the BTS ML consumer application, the BTS ML service and the message broker were deployed in the edge, whereas other services were in the cloud (Figure 4). Figure 6 shows the number of violations on three criteria: *ResponseTime*, *Data Quality*, and *Inference Accuracy*. We observe that the violation patterns of *Data Quality* and *Inference Accuracy* are fairly similar since the data quality always affects the prediction results. Taking a closer look at the quality monitoring in Figure 7, we can see that the *Inference Accuracy* drops every time the *Data Quality* falls down. In this case, the ML provider should set up suitable contracts and policies specifying at which level of data quality, the ML service would work properly to avoid *Inference Accuracy* violations.

In previous tests, the model was only trained with 80% random data collected from 0am to 2am and the data quality was examined using the data distribution in this period. Because the IoT data in different periods of a day may have various distributions, we decided to retrain our model every 2-hour using the most recent data. The violation rates, when evaluating our ML model with data from different periods, are recorded in Table 4. Since the model is trained periodically, the violation rates have been reduced significantly. With violation monitoring, we can perform further analysis and understand why the customer requirements are not satisfied during some period of service usage time, for example, the ML model is outdated for dealing with IoT data at the peak time of business – from 6am-10am is when the BTS usage is intensive due to active mobile users. Thereby, the ML service provider knows when the ML models need to be updated to sustain the quality of ML services, e.g., using domain knowledge (IoT data and telcos as a business sector). Given a comprehensive set of attributes and monitoring probes, we can learn from violations, discover their root causes, and improve ML services.

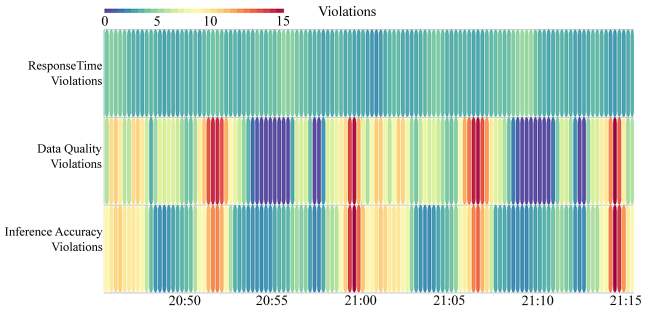

Figure 6. *ResponseTime, Data Quality, Inference Accuracy* violations

4.3. Benefits, flexibility and extensibility discussion

The obvious benefit for stakeholders is to establish agreed constraints and to monitor them at runtime. Different stakeholders carry out their own ways of monitoring and evaluation but they can use the results to discuss and explain unexpected behavior of the ML service. However, using our framework, we could move to the step of continuous testing and observability of ML production, like the Netflix method [21]: critical attributes like data quality, inference accuracy and bias can be continuously evaluated by also injecting suitable test data via designed test cases.

In the context of dynamic ML services, flexible contracts and policies are important. Our proposed ML contract framework consists of various attributes that can be implemented in different ways. Given contract constraints in *JSON*, the developer and provider can easily describe stakeholders requirements. Meanwhile, the contract policy is written in *Rego*, an easy-to-use declarative language. The decoupling between policies and attribute constraints enables flexibility in terms of monitoring and evaluation. The contract and policies can be updated using REST API during operation due to requirements and conditions changing. With JSON and Rego, we can also integrate with other complex policies seen nowadays in state-of-the-art virtualized environments for ML services. QoA4ML also offers a high reusable contract and policy. An ML contract can be reused by adjusting the values corresponding to the requirements and policies for each customer. Since ML services often have many attributes in common, the developer only has to adapt appropriate parts to create separate contracts for multiple services.

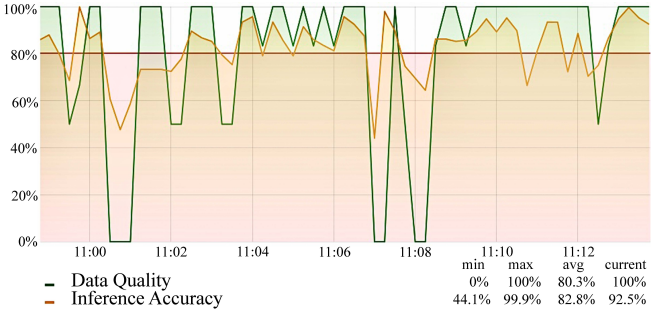

Figure 7. *Data Quality & Inference Accuracy* monitoring

Time Period	Violation Rate (training once)	Violation Rate (training every 2-hour)
0am-2am	2%	2%
2am-4am	4%	3%
4am-6am	18%	4%
6am-8am	15%	6%
8am-10am	10%	4%
10am-12pm	7%	4%
12pm-2pm	3%	3%
2pm-4pm	5%	5%
4pm-6pm	10%	7%
6pm-8pm	15%	5%
8pm-10pm	4%	2%
10pm-0am	1%	3%

TABLE 4. VIOLATION RATES *(Inference Accuracy)* WITH DIFFERENT TRAINING STRATEGIES

Frameworks	Types of service development	ML-specific attributes	ML contract specifications	ML contract monitoring
Google AI Platform [22]	ML models, services & infrastructures	Response time, training and serving cost for resource usage	No (common metrics for the ML infrastructure)	no (only resource monitoring)
Azure ML [23]	ML models, services & infrastructures	Inference accuracy (up to the developer); data drift	No ML specific contract (just for the infrastructure)	Fairness scores; model monitoring
Amazon Sager [24]	ML models, services and infrastructures	Self-specified and managed by developers but not included in service contract	No ML specific contract (pricing based on time, resource instance)	No (only resource monitoring)
Hopsworks [25]	ML services development and operation	Network security (TSL/SSL), data analysis & quality control	No specification for programming contracts	Monitoring tool but it is not part of contract
Algorithmia [26]	Framework for MLOps	Security, cost based on execution time	No specification for programming contracts	No (only resource and service monitoring)
Seldon [27]	Deploying ML models on Kubernetes	Data quality, runtime model management	Subscription plans	No (contract for SLA and service availability)
Hydrosphere [28]	Serving and monitoring for ML models	Outlier data score, data drift metrics, interpretability feature,	No contract	Monitoring of drift, model performance

TABLE 5. EXAMPLES OF SERVICE CONTRACTS IN SYSTEMS FOR ML DEVELOPMENT AND OPERATION

5. Related Work

ML service contracts: Service contracts have been researched intensively, e.g., [29], [30], [31], [32], but they do not support ML-specific attributes. In [4], important non-functional attributes for ML and their research questions are discussed. The work in [33] also discussed some ML-specific attributes related to the explainability and transparency in ML analysis. Our QoA4ML specification relies on existing identified metrics and attributes in these works and in extensive literature of ML. However, we introduce a specification for ML-specific attributes. Although the service contracts for ML are not clearly defined and developed, there exist many works for optimizing SLA for ML, such as Swayam [34], MArk [35], Clipper [36] and Clockwork [37]. Such optimization works are important and relevant to the implementation of contract enforcement, e.g., maximizing inference accuracy while minimizing costs and response time. Our work is related to ML optimization but, in this paper, we do not focus on ML optimization.

Supporting ML contracts development in ML systems: Many ML systems/platform-as-a-service support the development and operation of ML services. They include ML frameworks, supporting ML service DevOps and ML infrastructures, for example, Google Cloud AI [22], Microsoft Azure ML, Hopsworks [25], Algorithmia [26], Seldon [27] and Hydrosphere [28]. Most of them, together with offerings of ML infrastructures, enable features for obtaining and reporting common and ML-specific metrics, such as inference accuracy, response time, and cost. However, most of them do not have the notion of ML service contracts and the support for programming contracts. Table 5 shows a short summary of service contracts in existing development frameworks. Overall, our work differs from these frameworks by providing high-level contract support. In principle, our work can be integrated into these frameworks for the developer.

6. Conclusions and Future Work

Supporting ML contracts in ML services is important as many businesses develop and deploy ML models for consumers using existing third-party complex ML infrastructures. In this paper, we address ML-specific attributes and constraints in service contracts for MLaaS, focusing on ML serving for dynamic inference. We have proposed the QoA4ML framework to capture important ML-specific attributes for service contracts and to provide specifications for ML service providers and ML service consumers to consider part of their contracts. We implemented a prototype of QoA4ML and demonstrated how to support QoA4ML through monitoring and contract evaluations (the current prototype is available at https://github.com/rdsea/QoA4ML).

We have just experimented contracts between ML providers and consumers. For future experiments, we will need to deal with more complex situations when also considering the contracts between ML service provider and ML infrastructure provider, which mostly fall into the current cloud service contract model. We are working on extending ML attributes and advanced QoA4ML probes on accuracy, fairness and security. The future work will also be focused on combining QoA4ML with common contract specifications to provide a full-fledged service contract and optimization for MLaaS.

Acknowledgments

We are grateful to Tra-Anh Nguyen for her contribution and discussion in the business service contract of MLaaS and to My-Linh Nguyen for prototype contribution. The authors wish to acknowledge CSC IT Center for Science, Finland, for cloud resources.

References

[1] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[2] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. L. García, I. Heredia, P. Malík, and L. Hluchỳ, "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 77–124, 2019.

[3] R. Bianchini, M. Fontoura, E. Cortez, A. Bonde, A. Muzio, A.-M. Constantin, T. Moscibroda, G. Magalhaes, G. Bablani, and M. Russinovich, "Toward ml-centric cloud platforms," *Commun. ACM*, vol. 63, no. 2, p. 5059, Jan. 2020.

[4] J. Horkoff, "Non-functional requirements for machine learning: Challenges and new directions," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 386–391.

[5] D. M. W. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2020.

[6] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 2018, pp. 80–89.

[7] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Commun. ACM*, vol. 63, no. 1, p. 6877, Dec. 2019.

[8] C. Reed, E. Kennedy, and S. N. Silva, "Responsibility, autonomy and accountability: Legal liability for machine learning," Queen Mary School of Law Legal Studies Research Paper No. 243/2016, 2016. [Online]. Available: https://ssrn.com/abstract=2853462

[9] P. Hacker, R. Krestel, S. Grundmann, and F. Naumann, "Explainable AI under contract and tort law: legal incentives and technical challenges," *Artificial Intelligence and Law*, jan 2020.

[10] "Open Policy Agent," https://www.openpolicyagent.org, accessed: Nov 30, 2020.

[11] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Comput. Surv.*, vol. 54, no. 6, Jul. 2021.

[12] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. A. Patterson, H. Tang, G. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. M. Hazelwood, A. Hock, X. Huang, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, C. Wu, L. Xu, C. Young, and M. Zaharia, "Mlperf training benchmark," *CoRR*, vol. abs/1910.01500, 2019.

[13] L. Song and P. Mittal, "Systematic evaluation of privacy risks of machine learning models," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/song

[14] P. Hacker, R. Krestel, S. Grundmann, and F. Naumann, "Explainable ai under contract and tort law: legal incentives and technical challenges," *Artificial Intelligence and Law*, pp. 1 – 25, 2020.

[15] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark," *SIGOPS Oper. Syst. Rev.*, vol. 53, no. 1, pp. 14–25, Jul. 2019.

[16] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, "A taxonomy and terminology of adversarial machine learning," October 2019.

[17] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[18] "Prometheus," https://prometheus.io/, accessed: Dec 04, 2020.

[19] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.

[20] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[21] P. Alvaro, K. Andrus, C. Sanden, C. Rosenthal, A. Basiri, and L. Hochstein, "Automating failure testing research at internet scale," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1728. [Online]. Available: https://doi.org/10.1145/2987550.2987555

[22] "Google Cloud AI Platform," accessed: Nov 12, 2020. [Online]. Available: https://cloud.google.com/ai-platform

[23] "Azure Machine Learning," https://docs.microsoft.com/en-us/azure/machine-learning/, accessed: Dec 04, 2020.

[24] "Amazon SageMaker," https://aws.amazon.com/sagemaker/, accessed: Dec 04, 2020.

[25] Logical Clocks, "Hopsworks platform for data intensive ai - logical clocks," Accessed: 2020-11-12. [Online]. Available: https://www.logicalclocks.com/hopsworks

[26] "Algorithmia," accessed: Nov 12, 2020. [Online]. Available: https://algorithmia.com/

[27] "Seldon," accessed: Nov 12, 2020. [Online]. Available: https://www.seldon.io/

[28] "Hydrosphere," accessed: Nov 12, 2020. [Online]. Available: https://hydrosphere.io/

[29] D. Serrano, S. Bouchenak, Y. Kouki, F. A. [de Oliveira Jr.], T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "Sla guarantees for cloud services," *Future Generation Computer Systems*, vol. 54, pp. 233 – 246, 2016.

[30] R. B. Uriarte, F. Tiezzi, and R. De Nicola, "Slac: A formal service-level-agreement language for cloud computing," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 419–426.

[31] A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes, "Automating sla-driven api development with sla4oai," in *Service-Oriented Computing*. Springer International Publishing, 2019, pp. 20–35.

[32] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, *Service Level Agreements for Cloud Computing*. Springer Publishing Company, Incorporated, 2011.

[33] S. Oppold and M. Herschel, "A system framework for personalized and transparent data-driven decisions," in *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, vol. 12127. Springer, 2020, pp. 153–168.

[34] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: Distributed autoscaling to meet slas of machine learning inference services with resource efficiency," in *Middleware '17*, December 2017.

[35] C. Zhang, M. Yu, F. Yan *et al.*, "Enabling cost-effective, slo-aware machine learning inference serving on public cloud," *IEEE Transactions on Cloud Computing*, 2020.

[36] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 613–627.

[37] A. Gujarati, R. Karimi, S. Alzayat, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," *arXiv preprint arXiv:2006.02464*, 2020.