A Data Quality-Driven View of MLOps

Cedric Renggli[†] Luka Rimanic[†] Nezihe Merve Grel[†] Bojan Karlaš[†] Wentao Wu[‡] Ce Zhang[†]
[†]ETH Zurich
[‡]Microsoft Research
{cedric.renggli, luka.rimanic, nezihe.guerel, bojan.karlas, ce.zhang}@inf.ethz.ch
wentao.wu@microsoft.com

Abstract

Developing machine learning models can be seen as a process similar to the one established for traditional software development. A key difference between the two lies in the strong dependency between the quality of a machine learning model and the quality of the data used to train or perform evaluations. In this work, we demonstrate how different aspects of data quality propagate through various stages of machine learning development. By performing a joint analysis of the impact of well-known data quality dimensions and the downstream machine learning process, we show that different components of a typical MLOps pipeline can be efficiently designed, providing both a technical and theoretical perspective.

1 Introduction

A machine learning (ML) model is a software artifact "compiled" from data [?]. This point of view motivates a study of both similarities and distinctions when compared to traditional software. *Similar to* traditional software artifacts, an ML model deployed in production inevitably undergoes the DevOps process — a process whose aim is to "*shorten the system development life cycle and provide continuous delivery with high software quality*" [?]. The term "MLOps" is used when this DevOps process is specifically applied to ML [?]. *Different from* traditional software artifacts, the quality of an ML model (e.g., accuracy, fairness, and robustness) is often a reflection of the *quality of the underlying data*, e.g., noises, imbalances, and additional adversarial perturbations.

Therefore, one of the most promising ways to improve the accuracy, fairness, and robustness of an ML model is often to improve the dataset, via means such as data cleaning, integration, and label acquisition. As MLOps aims to *understand*, *measure*, and *improve* the quality of ML models, it is not surprising to see that *data quality* is playing a prominent and central role in MLOps. In fact, many researchers have conducted fascinating and seminal work around MLOps by looking into different aspects of data quality. Substantial effort has been made in the areas of data acquisition with weak supervision (e.g., Snorkel [?]), ML engineering pipelines (e.g., TFX [?]), data cleaning (e.g., ActiveClean [?]), data quality verification (e.g., Deequ [?, ?]), interaction (e.g., Northstar [?]), or fine-grained monitoring and improvement (e.g., Overton [?]), to name a few.

Meanwhile, for decades data quality has been an active and exciting research area led by the data management community [?, ?, ?], having in mind that the majority of the studies are agnostic to the downstream ML models (with prominent recent exceptions such as ActiveClean [?]). Independent of downstream ML models, researchers have studied different aspects of data quality that can naturally be split across the following four *dimensions* [?]: (1) *accuracy* – the extent to which the data are correct, reliable and certified for the task at hand; (2) *completeness* – the degree to which the given data collection includes data that describe the corresponding set of real-world objects; (3) *consistency* – the extent of violation of semantic rules defined over a set of data; and (4) *timeliness* (also referred to as *currency* or *volatility*) – the extent to which data are up-to-date for a task.

Table 1: Overview of our explorations with data quality propagation at different stages of an MLOps process.

Technical Problem for ML	**MLOps Stage**	**MLOps Question**	**Data Quality Dimensions**
Data Cleaning (Sec. 3) [**?**]	Pre Training	Which training sample to clean?	Accuracy & Completeness
Feasibility Study (Sec. 4) [**?**]	Pre Training	Is my target accuracy realistic?	Accuracy & Completeness
CI/CD (Sec. 5) [**?**]	Post Training	Am I overfitting to val/test?	Timeliness
Model Selection (Sec. 6) [**?**]	Post Training	Which samples should I label?	Completeness & Timeliness

Our Experiences and Opinions In this paper, we provide a bird's-eye view of some of our previous works that are related to enabling different functionalities with respect to MLOps. These works are inspired by our experience working hand-in-hand with academic and industrial users to build ML applications [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**], together with our effort of building `ease.ml` [**?**], a prototype system that defines an end-to-end MLOps process.

submissions/data-quality-ml-ops/figs/techq.pdf

Our key observation is that often *MLOps challenges are bound to data management challenges* — given the aforementioned strong dependency between the quality of ML models and the quality of data, the never-ending pursuit of *understanding, measuring*, and *improving the quality of ML models*, often hinges on *understanding, measuring*, and *improving the underlying data quality issues*. From a technical perspective, this poses unique challenges and opportunities. As we will see, we find it necessary to revisit decades of data quality research that are agnostic to downstream ML models and try to understand different data quality dimensions – accuracy, completeness, consistency, and timeliness – jointly with the downstream ML process.

In this paper, we describe four of such examples, originated from our previous research [**?**, **?**, **?**, **?**]. Table 1 summarizes these examples, each of which tackles one specific problem in MLOps and poses technical challenges of jointly analyzing data quality and downstream ML processes.

Outline In Section 2 we provide a setup for studying this topic, highlighting the importance of taking the underlying probability distribution into account. In Sections 3-6 we revisit components of different stages of the `ease.ml` system purely from a data quality perspective. Due to the nature of this paper, we avoid going into the details of the interactions between these components or their technical details. Finally, in Section 7 we describe a common limitation that all the components share, and motivate interesting future work in this area.

2 Machine Learning Preliminaries

In order to highlight the strong dependency between the data samples used to train or validate an ML model and its assumed underlying probability distribution, we start by giving a short primer on ML. In this paper we restrict ourselves on supervised learning in which, given a *feature space* $\mathcal{X}$ and a *label space* $\mathcal{Y}$, a user is given access to a dataset with n samples $\mathcal{D} := \{(x_i, y_i)\}_{i \in [n]}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Usually $\mathcal{X} \subset \mathbb{R}^d$, in which case a sample is simply a d-dimensional vector, whereas $\mathcal{Y}$ depends on the task at hand. For a regression task one usually takes $\mathcal{Y} = \mathbb{R}$, whilst for a classification task on C classes one usually assumes $\mathcal{Y} = \{1, 2, \ldots, C\}$.

We restrict ourselves to classification problems.

Supervised ML aims at *learning* a map $h : \mathcal{X} \to \mathcal{Y}$ that generalizes to unseen samples based on the provided labeled dataset $\mathcal{D}$. A common assumption used to learn the mapping is that all data points in $\mathcal{D}$ are sampled identically and independently (i.i.d.) from an unknown distribution $p(X, Y)$, where X, Y are random variables taking values in $\mathcal{X}$ and $\mathcal{Y}$, respectively. For a single realisation (x, y), we abbreviate $p(x, y) = p(X = x, Y = y)$.

The goal is to choose $h(\cdot) \in \mathcal{H}$, where $\mathcal{H}$ represents the hypothesis space, that minimizes the expected risk with respect to the underlying probability distribution [?]. In other words, one wants to construct h^* such that

$$h^* = \arg\min_{h \in \mathcal{H}} \mathbb{E}_{X,Y} \left(L(h(x), y) \right) = \arg\min_{h \in \mathcal{H}} \int_{\mathcal{X}} \int_{\mathcal{Y}} L(h(x), y) p(x, y) \, dy \, dx, \tag{1}$$

with $L(\hat{y}, y)$ being a loss function that penalizes wrongly predicted labels $\hat{y}$. For example, $L(\hat{y}, y) = \mathbf{1}(\hat{y} = y)$ represents the 0-1 loss, commonly chosen for classification problems. Finding the optimal mapping h^* is not feasible in practice: (1) the underlying probability $p(X, Y)$ is typically unknown and it can only be approximated using a finite number of samples, (2) even if the distribution were known, calculating the integral is intractable for many possible choices of $p(X, Y)$. Therefore, in practice one performs an empirical risk minimization (ERM) by solving $\hat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$. Despite the fact that the model is learned using a finite number of data samples, the ultimate goal is to learn a model which generalizes to any sample originating from the underlying probability distribution, by approximating its posterior $p(Y|X)$. Using $\hat{h}$ to approximate h^* can run into what-is-known as "overfitting" to the training set $\mathcal{D}$, which reduces the generalization property of the mapping. However, advances in statistical learning theory managed to considerably lower the expected risk for many real-world applications whilst avoiding overfitting [?, ?, ?]. Altogether, any aspect of data quality for ML application development should not only be treated with respect to the dataset $\mathcal{D}$ or individual data points therein, but also *with respect to the underlying probability distribution the dataset $\mathcal{D}$ is sampled from.*

Validation and Test Standard ML cookbooks suggest that the data should be represented by three disjoint sets to *train*, *validate*, and *test*. The validation set accuracy is typically used to choose the best possible set of hyper-parameters used by the model trained on the training set. The final accuracy and generalization properties are then evaluated on the test set. Following this, we use the term *validation* for evaluating models in the pre-training phase, and the term *testing* for evaluating models in the post-training phase.

Bayes Error Rate Given a probability distribution $p(X, Y)$, the lowest possible error rate achievable by *any* classifier is known in the literature as the Bayes Error Rate (BER). It can be written as

$$R^*_{X,Y} = \mathbb{E}_X \left[1 - \max_{y \in \mathcal{Y}} p(y|x) \right], \tag{2}$$

and the map $h_{opt}(x) = \arg\max_{y \in \mathcal{Y}} p(y|x)$ is called the *Bayes Optimal Classifier*. It is important to note that, even though h_{opt} is the best possible classifier (that is often intractable for the reason stated above), its expected risk might still be greater than zero, which results in the accuracy being at most $1 - R^*_{X,Y}$. In Section 4, we will outline multiple reasons and provide examples for a non-zero BER.

Concept Shift The general idea of ML described so far assumes that the probability distribution $P(X, Y)$ remains fixed over time, which is sometimes not the case in practice [?, ?, ?]. Any change of distribution over time is known as a *concept shift*. Furthermore, it is often assumed that both the feature space $\mathcal{X}$ and label space $\mathcal{Y}$ remain identical over a change of distribution, which could also be false in practice. A change of $\mathcal{X}$ or $p(X)$ (marginalized over Y) is often referred to as a *data drift*, which can result in missing values for training or evaluating a model. We will cover this specific aspect in Section 3. When a change in $p(X)$ modifies $p(Y|X)$, this is known as a *real drift* or a *model drift*, whereas when $p(Y|X)$ stays intact it is a *virtual* submissions/data-quality-ml-ops/figs/cpclean.png

Figure 1: Illustration of the relation between Certain Answers and Certain Predictions [?]. On the right, Q1 represents a *checking query*, whereas Q2 is a *counting query*.

drift. Fortunately, virtual drifts have little to no impact on the trained ML model, assuming one managed to successfully approximate the posterior probability distribution over the entire feature space $\mathcal{X}$.

3 MLOps Task 1: Effective ML Quality Optimization

One key operation in MLOps is seeking a way to improve the quality (e.g., accuracy) of a model. Apart from trying new architectures and models, improving the quality and quantity of the training data has been known to be at least as important [?, ?]. Among many other approaches, data cleaning [?], the practice of fixing or removing noisy and dirty samples, has been a well-known strategy for improving the quality of data.

MLOps Challenge When it comes to MLOps, a challenge is that not all noisy or dirty samples matter equally to the quality of the final ML model. In other words – when "propagating" through the ML training process, noise and uncertainty of different input samples might have vastly different effects. As a result, simply cleaning the input data artifacts either randomly or agnostic to the ML training process might lead to a sub-optimal improvement of the downstream ML model [**?**]. Since the cleaning task itself is often performed "semi-automatically" by human annotators, with guidance from automatic tools, the goal of a *successful* cleaning strategy from an MLOps perspective should be to minimize the amount of human effort. This typically leads to a partially cleaned dataset, with the property that cleaning additional training samples would not affect the outcome of the trained model (i.e., the predictions and accuracy on a validation set are maintained).

A Data Quality View A principled solution to the above challenge requires a *joint* analysis of the impact of incomplete and noisy data in the training set on the quality of an ML model trained over such a set. Multiple seminal works have studied this problem, e.g., ActiveClean [**?**]. Inspired by these, we introduced a principled framework called CPClean that models and analyzes such a noise propagation process together with principled cleaning algorithms based on sequential information maximization [**?**].

Our Approach: Cleaning with CPClean CPClean directly models the noise propagation — the noises and incompleteness introduce multiple possible datasets, called *possible worlds* in relational database theory, and the impact of these noises to final ML training is simply the *entropy* of training multiple ML models, one for each of these possible worlds. Intuitively, the smaller the entropy, the less impactful the input noise is to the downstream ML training process. Following this, we start by initiating all possible worlds (i.e., possible versions of the training data after cleaning) by applying multiple well-established cleaning rules and algorithms independently over missing feature values. CPClean then operates in multiple iterations. At each round, the framework suggests the training data to clean that minimizes the *conditional entropy* of possible worlds over the partially clean dataset. Once a training data sample is cleaned, it is replaced by its cleaned-up version in all possible worlds. At its core, it uses a *sequential information-maximization* algorithm that finds an approximate solution (to this NP-Hard problem) with theoretical guarantees [**?**]. Calculating such an entropy is often difficult, whereas in CPClean we provide efficient algorithms which can calculate this term in polynomial time for a specific family of classifiers, namely k-nearest-neighbour classifiers (kNN).

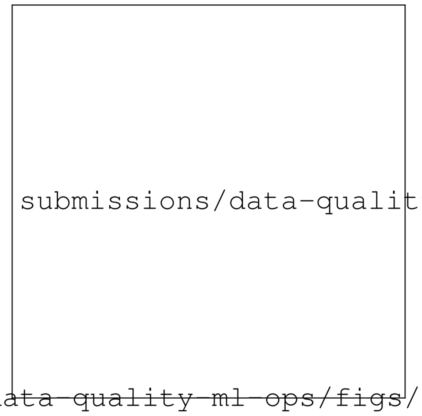
This notion of learning over incomplete data using *certain predictions* is inspired by research on *certain answers* over incomplete data [**?**, **?**, **?**]. In a nutshell, the latter reasons about *certainty* or *consistency* of the answer to a given input, which consists of a query and an incomplete dataset, by enumerating the results over all possible worlds. Extending this view of data incompleteness to non-relational operator (e.g., an ML model) is a natural yet non-trivial endeavor, and Figure 1 illustrates the connection.

Limitations Taking the downstream ML model into account for prioritizing human cleaning effort is not new. ActiveClean [**?**] suggests to use information about the *gradient* of a fixed model to solve this task. Alternatively, our framework relies on consistent predictions and, thus, works on an unlabeled validation set and on ML models that are not differentiable. In [**?**] we use kNN as a proxy to an arbitrary classifier, given its efficient implementation despite exponentially many possible worlds. However, it still remains to be seen how to extend this principled framework to other types of classifiers. Moreover, combining both approaches and supporting a labor-efficient cleaning approach for general ML models remains an open research problem.

4 MLOps Task 2: Preventing Unrealistic Expectations

In DevOps practices, new projects are typically initiated with a *feasibility study*, in order to evaluate and understand the probability of success. The goal of such a study is to prevent users with unrealistic expectations from submissions/data-quality-ml-ops/figs/ILSVRC2012_val_00006874.JPEG

(a) Image (#6874) illustrating a non-unique label probability.


submissions/data-quality-ml-ops/figs/ILSVRC2012_val_00004463.JPEG

(b) Image (#4463) showing multiple classes for a fixed sample.

submissions/data-quality-ml-ops/fi

(c) Image (#32040) mislabeled as a "pizza".

Figure 2: ImageNet examples from the validation set illustrating possible reasons for a non-zero Bayes Error.

spending a lot of of money and time on developing solutions that are doomed to fail. However, when it comes to MLOps practices, such a feasibility study step is largely missing — we often see users with high expectations, but with a very noisy dataset, starting an expensive training process which is almost surely doomed to fail.

MLOps Challenge One principled way to model the feasibility study problem for ML is to ask: *Given an ML task, defined by its training and validation sets, how to estimate the error that the best possible ML model can achieve, without running expensive ML training?* The answer to this question is linked to a traditional ML problem, i.e., to estimate the *Bayes error rate* (also called *irreducible error*). It is a quantity related to the underlying data distribution and estimating it using finite amount of data is known to be a notoriously hard problem. Despite decades of study [**?**, **?**, **?**], providing a practical BER estimator is still an open research problem and there are no known practical systems that can work on real-world large-scale datasets. One key challenge to make feasibility study a practical MLOps step is to understand how to utilize decades of theoretical studies on the BER estimation and which compromises and optimizations to perform.

Non-Zero Bayes Error and Data Quality Issues At the first glance, even understanding why the BER is not zero for every task can be quite mysterious — *if we have enough amount of data and a powerful ML model, what would stop us from achieving perfect accuracy?* The answer to this is deeply connected to data quality. There are two classical data quality dimensions that constitute the reasons for a non-zero BER: (1) *completeness* of the data, violated by an insufficient definition of either the feature space or label space, and (2) *accuracy* of the data, mirrored in the amount of noisy labels. On a purely mathematical level, the reason for a non-zero BER lies in *overlapping* posterior probabilities for different classes, given a realisable input feature. More intuitively, for a given sample the label might not be unique. In Figure 2 we illustrate some real-world examples from the validation set of *ImageNet* [**?**]. For instance, Figure 2a is labeled as a golfcart (n03445924) whereas there is a non-zero probability that the vehicle belongs to another category, for instance a tractor (n04465501) – additional features can resolve such an issue by providing more information and thus leading to a single possible label. Alternatively, there might in fact be multiple "true" labels for a given image. Figure 2b shows such an example, where the posterior of class rooster (n01514668) is equal to the posterior of the class peacock (n01806143), despite being only labeled as a rooster in the dataset – changing the task to a multi-label problem would resolve this issue. Finally, having noisy labels in the validation set yields another sufficient condition for a non-zero BER. Figure 2c shows such an example, where a pie is incorrectly labeled as a pizza (n07873807).

A Data Quality View There are two main challenges in building a practical BER estimator for ML models to characterize the impact of data quality to downstream ML models: (1) the computational requirements and (2) the choice of hyper-parameters. Having to estimate the BER in today's high-dimensional feature spaces requires a large amount of data in order to give a reasonable estimate in terms of accuracy, which results in a high computational cost. Furthermore, any practical estimator should be insensitive to different hyper-parameters, as no information about the data or its underlying distribution is known *prior to* running the feasibility study.

Our Approach: `ease.ml/snoopy` We design a novel BER estimation method that (1) has no hyper-parameters to tune, as it is based on nearest-neighbor estimators, which are non-parametric; and (2) uses pre-trained embeddings, from public sources such as PyTorch Hub or Tensorflow Hub[1], to considerably decrease the dimension of the feature space. The aforementioned functionality of performing a feasibility study using `ease.ml/snoopy` is illustrated in the above figure. For more details we refer interested readers to both the full paper [**?**] and the demo paper for this component [**?**]. The usefulness and practicality of this novel approach is evaluated on well-studied standard ML benchmarks through a new evaluation methodology that injects label noise of various amounts and follows the evolution of the BER [**?**]. It relies on our theoretical work [**?**], in which we furthermore provide an in-depth explanation for the behavior of kNN over (possibly pre-trained) feature transformations by showing a clear trade-off between the increase of the BER and the boost in convergence speed that a transformation can yield.

`submissions/data-quality-ml-ops/figs/snoopy.p`

Limitations The standard definition of the BER assumes that both the training and validation data are drawn i.i.d. from the *same* distribution, an assumption that does not always hold in practice. Extending our work to a setup that takes into account two different distributions for training and validation data, for instance as a direct consequence of applying data programming or weak supervision techniques [**?**], offers an interesting line of future research, together with developing even more practical BER estimators for the i.i.d. case.

[1] `https://pytorch.org/hub` and `https://tfhub.dev`

5 MLOps Task 3: Rigorous Model Testing Against Overfitting

One of the major advances in running fast and robust cycles in the software development process is known as continuous integration (CI) [**?**]. The core idea is to carefully define and run a set of conditions in the form of tests that the software needs to successfully pass every time prior to being pushed into production. This ensures the robustness of the system and prevents unexpected failures of production code even when being updated. However, when it comes to MLOps, the traditional way of reusing the same test cases repeatedly can introduce serious risk of overfitting, thus compromise the test result.

MLOps Challenge In order to generalize to the unknown underlying probability distribution, when training an ML model, one has to be careful not to overfit to the (finite) training dataset. However, much less attention has been devoted to the statistical generalization properties of the *test set*. Following best ML practices, the ultimate testing phase of a new ML model should either be executed only once per test set, or has to be completely obfuscated from the developer. Handling the test set in one way or the other ensures that no information of the test set is *leaked* to the developer, hence preventing potential overfitting. Unfortunately, in ML development environments it is often impractical to implement either of these two approaches.

A Data Quality View Adopting the idea of continuously testing and integrating ML models in productions has two major caveats: (1) test results are inherently random, due to the nature of ML tasks and models, and (2) revealing the outcome of a test to the developer could mislead them into overfitting towards the test set. The first aspect can be tackled by using well-established concentration bounds known from the theory of statistics. To deal with the second aspect, which we refer to as the *timeliness* property of testing data, there is an approach pioneered by Ladder [**?**], together with the general area of *adaptive analytics* (cf. [**?**]), that enable multiple reuses of the same test set with feedback to the developers. The key insight of this line of work is that the *statistical power* of a fixed dataset shrinks when increasing the number of times it is reused. In other words, requiring a minimum statistically-sound confidence in the generalization properties of a finite dataset limits the number of times that it can be reused in practice.

Our Approach: Continuous Integration of ML Models with `ease.ml/ci` As part of the `ease.ml` pipeline, we designed a CI engine to address both aforementioned challenges. The workflow of the system is summarized in Figure 3. The key ingredients of our system lie in (a) the syntax and semantics of the test conditions and how to accurately evaluate them, and (b) an optimized *sample-size estimator* that yields a budget of test set re-uses before it needs to be refreshed. For a full description of the workflow as well as advanced system optimizations deployed in our engine, we refer the reader to our initial paper [**?**] and the followup work [**?**], which further discusses the integration into existing software development ecosystems.

We next outline the key technical details falling under the general area of ensuring generalization properties of finite data used to test the accuracy of a trained ML model repetitively.

Test Condition Specifications A key difference between classical CI test conditions and testing ML models lies in the fact that the test outcome of any CI for ML engine is inherently probabilistic. Therefore, when evaluating result of a test condition that we call a *score*, one has to define the desired confidence level and tolerance as an (ϵ, δ) requirement. Here ϵ (e.g., 1%) indicates the size of the confidence interval in which the estimated score has to lie with probability at least $1 - \delta$ (e.g., 99%). For instance, the condition `n - o > 0.02 +/- 0.01` requires that the new model is at least 2 points better in accuracy than the old one, with a confidence interval of 1 point. Our system additionally supports the variable `d` that captures the fraction of different predictions between the new and old model. For testing whether a test condition passes or fails one needs to distinguish two scenarios. On one hand, if the score lies outside the confidence interval (e.g., `n - o > 0.03` or `n - o < 0.01`), submissions/data-quality-ml-ops/figs/ci.png

Figure 3: The workflow of `ease.ml/ci`, our CI/CD engine for ML models [**?**].

the test immediately passes or fails. On the other hand, the outcome is ill-defined if the score lies inside the confidence interval. Depending on the task, user can choose to allow *false positive* or *false negative* results (also known as "type I" and "type II" errors in statistical hypothesis testing), after which all the scores lying inside the confidence interval will be automatically rejected or accepted.

Test Set Re-Uses In the case of a non-adaptive scenario in which no information is revealed to the developer after running the test, the least amount of samples needed to perform H evaluations with the same dataset is the same as running a single evaluation with δ/H error probability, since the H models are independent. Therefore, revealing any kind of information to the developer would result in a dataset of size H multiplied by the number of samples required for one evaluation with δ/H error. However, this trivial strategy is very costly and usually impractical. The general design of our system offers a different approach that significantly reduces the amount of test samples needed for using the same test set multiple times. More precisely, after every commit the system only reveals a binary pass/fail signal to the developer. Therefore, there are 2^H different possible sequences of pass/fail responses, which yields that the number of samples needed for H iterations is the same as running a single iteration with $\delta/2^H$ error probability – much smaller than the previous δ/H one. We remark that further optimizations can be deployed by making use of the structure or low variance properties that are present in certain test conditions, for which we refer the interested readers to the full paper [**?**].

Limitations The main limitation consists of the worst-case analysis which happens when the developer acts as an adversarial player that aims to overfit towards the hidden test set. Pursuing other, less pragmatic approaches to model the behavior of developers could enable further optimization to reduce the number of test samples needed in this case. A second limitation lies in the lack of ability to handle concept shifts. Monitoring a concept shift could be thought of as a similar process of CI – instead of fixing the test set and testing multiple models, one could fix a single model and test its generalization over multiple test sets. From that perspective, we hope that some of the optimizations that we have derived in our work could potentially be applied to monitoring concept shifts as well. Nevertheless, this needs further study and forms an interesting research path for the future.

6 MLOps Task 4: Efficient Continuous Quality Testing

One of the key motivations for DevOps principles in the first place is the ability to perform fast cycles and continuously ensure the robustness of a system by quickly adapting to changes. At the same time, both are well-known requirements from traditional software development that naturally extend to the MLOps world. One challenge faced by many MLOps practitioners is the necessity to deal with the shift of data distributions when models are in production. When new production data comes from a different (unknown) distribution, models trained over previously seen data distributions might not perform well anymore.

MLOps Challenge While there has been various research on automatic domain adaption [**?**, **?**, **?**], we identify a different challenge when presented with a collection of models, each of which could be a "staled model" or an automatically adapted model given some domain adaption method. This scenario is quite common in many companies — they often train distinct models on different slices of data independently (for instance one model for each season) and automatically adapt each of these models using different methods for new data. As a result, they often have access to a large set of models that could be deployed, hoping to know which one to use given a fresh collection of production data (e.g., the current time period such as the current day). The challenge is, given an unlabeled production data stream, to pick the model that performs best. From the MLOps perspective, the goal is to minimize the amount of labels needed to acquire in order to make such a distinction.

A Data Quality View Concept shift is by its definition related to the *timeliness* properties of the data. The available pre-trained models are intended to capture the changes of training data over time. Naturally, simple rules can be applied to choose the current model if one has access to some meta information about both the current timestamp and the pre-trained models (e.g., the current weekday and for each model the day it represents). This problem gets particularly difficult when there are no such meta-data available. In that case, having access to a fully labeled clean dataset would result in trivially selecting the pre-trained model that achieves the highest accuracy. Collecting labels for a large enough test set is very costly in practice compared to simply gathering a set of unlabeled samples though. The reason is that accurately labeling samples requires human, if not expert level, annotators. Consequently, one wishes to robustly solve the problem of *picking* the best model for the current time span with the fewest amount of labeling effort necessary and thus relying on *incomplete* test data with respect to their labels.

Our Approach: `ease.ml/ModelPicker` Model Picker is an online model selection approach to selectively sample instances that are informative for ranking pre-trained models [**?**]. Specifically, given a set of pre-trained models and a stream of unlabeled data samples that arrive sequentially from a data source, the Model Picker algorithm answers when to query the label of an instance, in order to pick the best model under limited labeling budget. We conduct a rigorous theoretical analysis to show that Model Picker has no regret for adversarial streams (e.g., non-i.i.d. data), and is effective in online prediction tasks for both adversarial and stochastic streams. Moreover, our theoretical bounds match (up to constants) those of existing online algorithms that have access to all the labels.

Limitations One immediate extension of Model Picker is towards a setting in which the user at once has access to a pool of unlabeled data samples. In such a *pool-based sampling* case [**?**], one can rank the entire collection of data samples to select the most informative example instead of scanning through the data sequentially to decide whether to query a label or not. Despite the applicability of Model Picker to such a scenario where one can form a stream by sampling i.i.d. from the pool of samples, the availability of entire data collection can be exploited to further reduce the annotation costs with a more specialized strategy for pool-based scenarios.

7 Moving Forward

We have briefly described four of our previous works with a unified theme — all of them provide, in our opinion, *functionalities that are useful to facilitate a better MLOps process*, which, on the flip side, introduce new fundamental technical problems that require us to *jointly analyze the impact of data quality issues to downstream ML processes.* When studying these technical problems, we often need to go beyond an ML-agnostic view of data quality and, instead, need to develop new methods that *simultaneously* combine the two aspects of ML *and* data quality. Despite the progress that we have made so far, this endeavor is still at its early stages. In the following, we present two future directions that, in our opinion, are necessary to facilitate both MLOps as an important functionality and ML-aware data quality as a fundamental research area.

ML-Aware Data Quality From a technical perspective, jointly understanding data quality and downstream ML processes is both interesting and challenging. All results we discussed in this paper are arguably limited [**?**, **?**, **?**, **?**] — after starting from a principled formulation of a problem, reaching fundamental computational challenges within these principled frameworks is inevitable. We get around those by either (1) opting for simpler proxy models for which we can derive stronger results and/or more efficient algorithms (e.g., kNN used in `ease.ml/snoopy` [**?**] and CPClean [**?**]) or (2) optimizing for specific cases commonly used in practice (e.g., the patterns in `ease.ml/ci` [**?**] that we optimized for). To further facilitate MLOps in general, we are in dire need for an ML-aware data quality that is not only principled, but also practical for a larger collection of scenarios and ML models. These are all technically challenging — simply extending the methods that we developed is unlikely to succeed. We hope that our current endeavors [**?**, **?**, **?**, **?**] can serve, in some ways, as "examples of failures" that other researchers can draw inspirations from.

Beyond Accuracy Another common limitation of our work [**?**, **?**, **?**, **?**] is that they all focus on improving the *accuracy* of an ML model artifact. Although this is one of the most important aspects of model quality, recently researchers have also identified multiple interesting dimensions of model quality such as robustness, fairness, and explainability. Even though we expect these quality dimensions to become the core of the MLOps process in the future, how to extend functionalities that we developed for improving accuracy to these quality dimensions is still an open question. Jointly analyzing the impact of all data-quality dimensions with respect to more than a single metric that quantifies ML models is a large and promising research area that we believe will provide further understanding and improvements of the MLOps process.

Acknowledgments

We are grateful to many collaborators that we have been working with over the years, especially (in the context of this paper) those who also contributed to the development of techniques in [**?**, **?**, **?**, **?**] including Peng Li, Prof. Xu Chu, Mohammad Reza Karimi, and Prof. Andreas Krause. These technical results would not be possible without their contributions.

CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.