Bulletin of the Technical Committee on

Data Engineering

March 2021 Vol. 44 No. 1

IEEE Computer Society

Letters

Opinions

Special Issue on Data Validation for Machine Learning Models and Applications

Conference and Journal Notices

The TC on Data Engineering
Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is `http://tab.computer.org/tcde/index.html`.

The Data Engineering Bulletin
The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at `http://tab.computer.org/tcde/bull_about.html`.

Letter from the Editor-in-Chief

The March issue of the Data Engineering Bulletin focuses on the intricate interplay as well as a significant gap between data management and machine learning when it comes to supporting real-life business applications.

The opinion piece of this issue features a group of distinguished researchers and their assessment and prognosis of machine learning's current and future roles in building database systems. Besides highlighting several specific potentials and challenges such as using machine learning to optimize database indices and query optimization, the article also gives a great overview of how databases, data analytics and machine learning, system and infrastructure, work together to support today's business needs. It is clear that the business needs, the volume, velocity, and variety of the data, the latency and throughput requirements have evolved dramatically and in consequence, data management systems must adapt. The opinion pieces described four disruptive forces underneath the evolution, which are likely to influence future data systems.

Our associate editor Sebastian Schelter put together the current issue–Data Validation for Machine Learning Models and Applications–that consists of six papers from leading researchers in industry and academia. The papers focus on data validation, which is a critical component in end-to-end machine learning pipelines that many business applications rely on.

Haixun Wang
Instacart

Letter from the Special Issue Editor

Software applications that learn from data using machine learning (ML) are being deployed in increasing numbers in the real world. Designing and operating such applications introduces novel challenges, which are very different from the challenges encountered in traditional data processing scenarios. ML applications in the real world exhibit a much higher complexity than "text book" ML scenarios (e.g., training a classifier on a pre-existing dataset). They do not only have to learn a model, but must define and execute a whole ML pipeline, which includes data preprocessing operations such as data cleaning, standardisation and feature extraction in addition to learning the model, as well as methods for hyperparameter selection and model evaluation. Such ML pipelines are typically deployed in systems for end-to-end machine learning, which require the integration and validation of raw input data from various input sources, as well as infrastructure for deploying and serving the trained models. The system must also manage the lifecycle of data and models in such scenarios, as new (and potentially changing) input data has to be continuously processed, and the corresponding ML models have to be retrained and managed accordingly. The majority of these challenges have only recently begun to attract the attention of the data management community. A major obstacle is that the behavior of ML-based systems heavily depends on the consumed input data, which can rapidly change, for example due to changed user behavior or due to errors in external sources that produce the inputs. This area represents a gap between the data management and ML communities: research in ML mostly focuses on learning algorithms, and research in data management is mostly concerned with data processing and integration. In this issue, we focus on this gap in data validation for machine learning, and provide perspectives from both the academic and industrial research communities to learn about the state of the art, open problems and to uncover interesting research directions for the future.

The first paper presents *A Data Quality-Driven View of MLOps* and demonstrates how different aspects of data quality propagate through various stages of machine learning development. It connects data quality to the downstream machine learning process, an approach that is also taken by our second paper, which argues that we should move *From Cleaning before ML to Cleaning for ML*. The authors propose an end-to-end approach to take the entire application's semantics and user goals into account when cleaning data, instead of performing the cleaning operations in an isolated manner beforehand.

The next two papers on *Validating Data and Models in Continuous ML pipelines* and *Automated Data Validation in Machine Learning Systems* from Google and Amazon provide us with an industry perspective on the area in the focus of this issue. The first paper describes tools developed at Google for the analysis and validation of two of the most important types of artifacts: Datasets and Models. These tools (which are part of the Tensorflow Extended Platform) are currently deployed in production at Google and other large organizations, and are heavily inspired by well-known principles of data-management systems. The second paper from Amazon reviews some of the solutions developed to validate data at the various stages of a data pipeline in modern ML applications, discusses to what extent these solutions are being used in practice, and outlines research directions for the automation of data validation.

The subsequent paper on *Enhancing the Interactivity of Dataframe Queries by Leveraging Think Time* focuses on the highly exploratory and iterative nature of data validation in the early stages of an ML application, where data scientists start with a limited understanding of the data content and quality, and perform data validation through incremental trial-and-error. The final paper of this issue on *Responsible AI Challenges in End-to-end Machine Learning* completes the view on data validation for machine learning by connecting it with pressing issues from the area of responsible data management.

Working on this issue has been a privilege for me, and I would like to thank the authors for their contributions.

Sebastian Schelter
University of Amsterdam & Ahold Delhaize Research, Netherlands

ML-In-Databases: Assessment and Prognosis

Tim Kraska, Umar Farooq Minhas, Thomas Neumann, Olga Papaemmanouil, Jignesh M. Patel, Chris Ré, Michael Stonebraker

Abstract

With the rapid adoption of Machine Learning (ML) in Computing, there has been a flurry of recent research considering using ML to build the internal component of database systems. While initial work in this area has shown interesting results, the jury is still out on whether these methods will replace existing methods. A group of researchers with opinions on both sides of this issue met to assess the state of this area and to formulate a plan for the next steps that would be needed to determine the potential role of these new ML-based methods in building future database systems. This article summarizes the collective perspectives that resulted from these discussions. First, this article describes broad forces that are changing the landscape in which database systems are deployed, connecting several trends that likely require rethinking how future database engines are built. Next, this article describes the different perspectives on this topic of using ML methods to replace existing internal database components. Finally, the key takeaways from this discussion are presented, and these takeaways also point to directions for future research.

1 Introduction

Enterprises increasingly want to move to making data-driven decisions across every aspect of their business. To achieve this goal, they are increasingly adopting a rich ecosystem of data management tools, and database systems are invariably at the heart of these data ecosystems. However, we see four, possibly very disruptive forces, that are likely to strongly influence future data systems. These forces, which range from well-established ones (force #4) to some in their infancy (forces #1 and #2), are described next.

1.1 Force #1: The changing end-user landscape

First, from the end-user perspective, there is a clear goal in enterprises to democratize data, moving to holistic data-driven decision making across the entire enterprise. Essentially, enterprises want *every* business user in their organization to make data-driven decisions, and often in real time.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This transformation will require that a broad range of employees in an organization must have access to the underlying enterprise data (with appropriate access controls and often via applications). In this new world, the end user is a non-programmer business user, as opposed to a business analyst, who asks ad hoc questions using a variety of querying paradigms, including natural language and no-code interfaces.

With this driving force, the "audience" for a database system will explode from database programmers and business analysts to a much larger group of business users. This driving force will also change the technical programming skill set that one can expect from an end user.

1.2 Force #2: The changing application surface

Second, in this new data-democratized world that we see coming, end users will demand that the underlying platform carry out a broader range of data analysis tasks. These tasks go beyond traditional structured query processing (SQL) to include machine learning (both exploratory and predictive) and more complex analytics, such as regressions and principal components analysis. Moreover, we expect sophisticated visualizations and natural language processing (NLP) to become much more important. Further, these tasks must be carried out efficiently on an ever-increasing volume of data.

In addition, when deploying machine learning (ML) for prediction, there is a critical need (increasingly driven by regulations) to manage and archive the entire data pipeline that is associated with a deployed ML model. In other words, there is a need for data platforms to provide robust model management as an integrated core feature of the data platform itself.

1.3 Force #3: The changing data organization

Third, an often-quoted anecdote is that data scientists spend at least 80% of their time finding data sets they want to analyze, cleaning the data and integrating it together. A further anecdote is that independently constructed data sets never have a common schema, which makes the data integration problem noted above especially difficult. Obviously, analyses whether through ML or other means will produce garbage on dirty or inconsistent data. As a result, a key ground stake for this new data-democratized reality is that there must be a single source of truth for data in the entire enterprise. This aspect is leading to the consolidation of data spread across data silos into a common data repository, often called a data lake. To fully leverage this data lake, we see enterprises making substantially greater investments in data cataloging, data integration and data cleaning off into the future.

1.4 Force #4: The changing hardware substrate

The last driving force, and in many ways the force that is accelerating the other three forces described above, is the move to the cloud. This force fundamentally shifts the substrate on which analytic database software runs. Gone are the days in which a business intelligence database system ran on hardware consisting of a physical server box resident in the enterprise, and consisting of predefined fixed amounts of compute, memory, storage and networking components. In the cloud the server becomes a collection of disaggregated compute, memory, storage (with many more tiers), and networking resources, all of which can be mixed in just about any proportion.

Further, this virtual server can be elastically sized, often in a matter of seconds, to add or subtract any of these hardware resources. While such a virtual server may be a collection of many physical components spread across a physical data center (or data centers), logically, it presents itself as a single server. We call this virtual server a *cloud server* in the remainder of this article.

What is even more powerful is that cloud servers are more economical than traditional servers, as they are priced using a pay-as-you-go paradigm. Enterprises can size up their cloud server capacity for a short amount of time and size it down when done, making internal capital allocation processes far cheaper. Thus, large data projects now become more financially viable.

Traditional database systems are notorious for having a large number of tuning knobs that have to be continually tweaked for performance by human database administrators (DBAs). These knobs have to be set at the “right” values when the database is initially installed, and then re-tuned as the workload changes. This reliance on humans to keep database installations running efficiently often results in sub-optimal performance, costing enterprises time and money. Modern cloud data platforms take this problem head-on and are moving toward self-tuning. Thus, in the cloud era, there will be a huge reduction in the human IT cost associated with running a cloud data warehouse.

We see three consequences of these forces, to which we now turn.

2 Consequences of these forces

The database community has started to make fundamental changes to react and leverage the four driving forces described above. A key direction in recent years has been to rethink the methods that go into every internal component inside a database engine. These components include storage management (which includes indexing), query processing methods, query optimization, and query scheduling. In a traditional data processing system these components are tightly coupled at the architecture level and optimized to run on a specific hardware configuration.

In a cloud server, the storage and compute is disaggregated in the underlying cloud fabric. Each of these components can be increased elastically independent of each other. Fast networking is often implicitly provided by the cloud provider. Further, each component of the storage hierarchy can be provisioned (and adjusted elastically). Thus, the bulk of the data sits in a cloud file system, which in principle has infinite storage capacity. That data can be accessed by compute resources by pulling data from the cloud file system (CFS). Along the way the data can be cached in multiple storage locations, including local SSDs and main memory. By-and-large traditional memory hierarchy rules apply. Storage that is closer to the processor is accessible at a lower latency, but has a higher unit storage cost, and lower capacity compared to storage that is “further away.” What is interesting however, is that in a cloud server, every layer of the storage hierarchy can be elastically adjusted. Essentially a data platform can now provision a virtual cloud server in an arbitrary configuration to best suit the target application needs, and then reconfigured easily (and economically) as the workload changes. Database systems that don’t leverage this elastic behavior, will simply not survive the competition in the cloud era.

Traditional database systems will need major reworking to achieve this sort of elasticity. Since a cloud server is assembled from a set of disaggregated underlying hardware resources, to make full use of this hardware organization, the internals of database engines must also be disaggregated to run at the speed of the underlying hardware. There is broad consensus in the database community that database internal components and architecture must be rethought from scratch for this new cloud server world.

Another trend which follows from these forces is the Software 2.0 movement [9]. This new approach to software engineering aims to move from thinking about software as a predefined static organization of code to a dynamic model-based view. In this new approach, machine learning models, often a neural network, are a fundamental software building block. This new building block can be viewed as a dynamic function that takes data as input, runs it through the ML model, and outputs the predictions from the model as the output/result of the function. A critical aspect of this new approach, which we call *ML-functions* (*MLF*s), is that an MLF can easily be retrained using new data. Thus, an MLF can dynamically optimize itself to adapt to new workload characteristics. This adaptation can often be done internally in the MLF providing an elegant software abstraction in which the software system can be considered to be a collection of dynamic, self-optimizing components. MLFs provide a disaggregated approach to software engineering and using it purposefully in this manner provides an interesting parallel to the disaggregation of hardware components in a cloud server. We expect complex software systems (not just database systems) will move toward this paradigm to reap the benefits of self-optimization.

Synergistic with the Software 2.0 approach is the move to serverless computing. With serverless computing, provisioning of hardware resources is automatically done by the serverless infrastructure freeing the developer from having to worry about that aspect of application development. Software 2.0 provides a new software abstraction while serverless provides a new hardware abstraction, and together both make the life of a modern software developer easier.

In summary, we are already seeing DBMSs being rearchitected for cloud deployment. In addition, serverless computing, such as AWS Lambda, are becoming an important cloud platform. Finally, the benefits of self-adaptation are obvious, which will drive large software systems, including DBMSs, toward using ML.

3 ML-in-Databases

Against the backdrop of changes in the broader data landscape described above, a panel was organized at the 2021 Annual Conference on Innovative Data Systems Research (CIDR). The topic for this panel was the role of Machine Learning (ML) in driving transformative changes to database internals. Below we summarize the discussion that started at the panel and continued after the initial panel discussion, resulting in the discussion and direction that is described in this article.

3.1 Learned indices

The first topic for discussion was the role of learned methods to reimagine a variety of database internal components. There has been a flurry of work over the last few years advocating the use of ML models to replace the traditional database internal components. This line of work can be considered as advocating for a Software 2.0 approach to reimagining database internal components. There have been two big themes in the community, namely learned indices and learned query optimizers. We discuss learned indices in this section, and we cover learned query optimization in the next section.

The learned index approach challenges the way in which database systems use indices. The initial focus of this line of research (e.g., [10, 7]) has been to provide an alternative to the "ubiquitous" B+-tree index structure [5]. In addition, there has been early work on other index structures, including LSMs and spatial indices. To focus this article, we only discuss the work related to the B+-tree index.

In a traditional database system, search on a large dataset can be sped up by building a B+-tree index. A B+-tree index is a disk-optimized balanced tree data structure that has a $log(N)$ average case search time and I/O complexity. However, one can view a B+-tree search operation as a lookup function that takes as input an argument (the search key) and returns the set of matching records from the underlying dataset as a result. In a B+-Tree index, this function has a large data structure (the index) that is associated with the search function. A learned index on the other hand can be viewed as replacing this data structure with a MLF, and often a hierarchical collection of MLFs.

The panel discussion on this topic focused on past approaches to learned B+-tree index structures and quickly pivoted to the central question: *Do these learned indices actually outperform traditional B+-trees, especially when one uses optimized B+-tree indices?*

Over the last four decades, a number of B+-tree optimizations have been proposed, including cache-efficient implementations, and use of sophisticated key compression methods. It was pointed out that both prefix and postfix compression methods have been used in VSAM indices [3], which have been in commercial systems for over four decades. These optimizations often produce huge efficiency improvements. Thus, comparison with an optimized B+-tree implementation that also includes considerations, such as a broad range of compression techniques, is needed to better understand the role of learned indices in practice.

It was also pointed out that while the research enthusiasm has been high for learned indices, practical adoption will take time and the final judgement about their practicality is still out. The early work on learned indices has largely ignored issues of concurrency control mechanisms, and non-main memory storage. Indices are typically common in transactional workloads and robust concurrency control methods are ground stakes for commercial adoption. There is only preliminary work to-date on how to make learned index structure work with a mix of update and read-only queries, and comparison with non-main memory database systems is missing. Further, the issue of crash recovery has not been addressed by work on learned indices, and that too is crucial for commercial traction.

The proponents of learned indices argued that comparisons with highly optimized B+-trees for in-memory settings including prefix compression (e.g., against the ART index) have been carried out [10, 12] but that other settings (disk, concurrency) are still missing. However, one should view the work on learned indices as work in progress.

The proponents of learned indices also pointed out that a critical missing component is the lack of realistic workloads, which is a broader community challenge. For example, if the data being indexed is Gaussian distributed, then a MLF will outperform a traditional index. Thus, the performance tradeoffs are very dependent on the distribution characteristics of the search keys. Getting realistic data distributions has been challenging, and this aspect is critical in determining the pros and cons of the two approaches to indexing data.

On the issue of performance of learned indices, there is a change in emphasis in parts of the learned index community from targeting execution time performance improvements to targeting space efficiency. Early results indicate that the space complexity of learned indices is provably better than B+-tree indices [8], and some commercial systems (e.g., Google Bigtable) observe throughput improvements because of the smaller index size [1]. Similarly, the smaller (and faster) lookup performance of learned indexes improved the end-to-end throughput of LSMs, like RocksDB [6]. However, the practical benefits depend on the workload, making good workloads/benchmark central to also moving this aspect of the learned index research forward.

3.2 Learned optimizers

Learned query optimizer is another dominant contemporary research theme in the database community. This sub-community proposes an learned alternative to traditional query optimization (e.g., [11, 14]). A traditional query optimizer takes as input a query statement (often an algebraic expression) and transforms it to an optimal plan (another annotated algebraic expression). The internal code organization of a traditional query optimizer is often a large mesh of functions (with some functions being rules-based) that work together to formulate an optimal execution plan.

Query optimization is a notoriously difficult task and one that after five decades of work continues to drive significant research attention. A learned approach to query optimization takes a Software 2.0 approach to this internal component of a database engine, and aims to use an MLF, often backed by a deep neural network, to optimize queries.

Similar to the discussion above on learned indices, it was pointed out that work on learned query optimizers is also preliminary, and there isn't conclusive evidence that such approaches are yet practical for real deployments. Besides, query optimization is really critical in analytic environments, and state-of-the-art systems use a column store organization in such settings, as column stores result in much better query performance. However, until recently all comparisons of learned query optimizers were against the PostgreSQL optimizer or traditional row-oriented optimizers (e.g., Oracle or Microsoft SQL Server). Showing gains for analytic queries on a row-store database does not make a strong case for learned query optimizers.

Proponents of the learned methods pointed out that existing work already compares against some of the best-known optimizers, such as Microsoft SQL Server and Oracle [14, 13], and noted that PostgreSQL is widely used as it is open-sourced. This open-source aspect is important as the research in this area requires making changes to the actual optimizer source code to implement the learned optimization methods. It was further noted that commercial systems have started to integrate learned cardinality estimates and other ML-based improvements.

Proponents of the learned methods also highlighted that there is an additional benefit of pursuing research in learned methods, which is to explore it as a research direction to determine what its boundaries are. Even if at the end of the day, they do not end up outperforming state-of-the-art systems, we will as a community learn a lot from this process. Thus, there is a pure research exploration component to pursuing learned approaches.

3.3 Instance optimality

The diversity of database workloads and the elastic nature of the cloud server on which a database instance runs, presents unique opportunities to tune a specific database instance to the workload and hardware parameters available at that instant. This idea of instance optimal database deployments is an emerging research trend in the database community. ML methods are clearly applicable here as many database systems have hundreds if not thousands of parameters, and the choice of the parameters plays a big role in determining the overall system performance. Further, even if the parameters have been set once, as the workload changes these parameters must be reset.

Today such database tuning is largely done statically. However, there is a growing body of research work in our community that advocates taking as input data from the operational environment – think database and system logs – and using it for self-tuning. There has been sporadic work over the last four decades on self-tuning, and ML-based algorithms are likely to play a role in this endeavor, if they can be shown to be superior to traditional methods.

3.4 ML-in-Databases, the "other" ML

This panel of researchers also noted that perhaps the real reason to bring ML methods into database engines is orthogonal to replacing existing internal database data structures/algorithms with learned parts. This reason is related to the rapidly changing database workloads and user base (recall Forces #1 and #2).

Database workloads are changing quite rapidly. With a unified data-centric view of the enterprise (Force #3) enterprises can run both analytic and ML workloads in the same data engine. Today, these two types of workloads are often run in two different platforms. For example, a platform that runs SQL analytics and another that runs ML analytics (e.g., Pandas or Tensorflow). Managing two platforms instead of one increases the data management costs and also makes data governance more complicated. This current approach of "ML-outside-the-database" is a huge and growing issue for enterprises.

The community has worked on the integration of ML and SQL in the past. Some commercial database systems (e.g., [4]) have been shipping ML methods that run inside the database engine since the earlier part of this century. However, what is needed is a dramatic new approach to not just make ML and SQL workloads operate in the same data engine, but also to support ML model management in the data platform. Today, there is an unmet need for robust management of ML pipelines (simplistically think of this as an "execution plan" with data cleaning, encoding, training, and scoring "operators"). To use an ML model for inferencing (e.g., making a credit recommendation for a new loan applicant), the enterprise has to put in place a vast data management infrastructure for the ML models. Regulatory or internal governance policies may require that the entire pipeline associated with creating the deployed ML model be archived so that it can be revisited in the future (e.g., to check for bias). As the underlying data changes, a previously optimal model may "drift," and may also need to be retrained to make more accurate inferences, which again requires support for robust model management.

In addition, if one looks at the life cycle of data, especially when an enterprise is trying to bring all data together in an enterprise-wide data lake (Force #2), one sees that there is ML and SQL intertwined at every step. Just getting the data into the data lake requires data cleaning, which will require using ML methods at scale. Further, this data cleaning "workload" is not a one-time job, but it is a workload that must be run continually. Users (and now there is a broader user base due to Force #1) also invoke analytics functions that often use ML for data exploration and visual discovery. In fact, in many enterprise data settings, these new data workloads (data cleaning and discovery) take up most of the time of the human user as well as most of the resources in the cloud server. When taking a holistic look at the workload present in a modern data ecosystem, the ML part is more dominant than traditional SQL.

Thus, the real opportunity for ML and database systems is to bring both ML processing and SQL processing together into the database engine. To achieve this goal, we need to rethink the internal organization of a database engine even more, and not just try to use ML/learned methods to speed up a traditional database platform, which arguably only supports SQL and thus a small fraction of the overall enterprise data workload.

Overall, a strong sentiment expressed in the panel was that the real synergies for ML and database to come together may lie in this area – namely, creating new data technologies that efficiently serve both workloads natively, and also provides in-built model management capabilities.

4 Key Takeaways

The discussion on this topic of ML-in-databases can be summarized by the following key takeaways, which also points to directions for future work.

Takeaway #1: The initial comparisons of leaned indices with optimized traditional indices should be further expanded to include concurrency control and multi-user settings. Learned indices need to make a holistic argument if they want to challenge the use of traditional indices.

Takeaway #2: A key benefit of learned indices may come from the learned structures requiring lower space utilization, rather than a reduction in search time. Further work in this area is needed to test this claim.

Takeaway #3: More realistic benchmarks/workloads are critical as the benefits of learned indices is very dependent on the data distribution. Coupled with Takeaway #1, it may be time for leaned indices to use end-to-end benchmarks in their comparison, with full parity on features like concurrency control, recovery, non main memory, and multi-user settings.

Takeaway #4: When evaluating the benefits of a learned optimizer, the yardstick for comparison should not just be PostgreSQL – a row-store system that is known to have low performance. Since access to the database system source code is required for this research, using an open-source column-store database system, such as MonetDB [2], is advocated.

Takeaway #5: To fully exploit the disaggregated and elastic nature of a cloud server, database deployments will need to consider instance optimal methods so that they can self-tune (and retune) themselves for the workload at hand. ML methods are likely critical here as long as they can demonstrate superiority to traditional methods.

Takeaway #6: The big opportunity for ML-in-Databases is for database systems to consider new data processing techniques/architectures that can efficiently process both ML and SQL tasks. The internal data structures used in such a platform may still be "traditional," but the huge opportunity ahead for the community is to expand the scope of workloads in this manner, and to include model management as a core data management task.

5 Acknowledgments

This work is supported in part (for effort by JMP) by NSF under grants OAC-1835446, IIS-1815701, and by CRISP – one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Hussam Abu-Libdeh, Deniz Altinbüken, Alex Beutel, Ed H. Chi, Lyric Doshi, Tim Kraska, Xiaozhou Li, Andy Ly, and Christopher Olston. Learned indexes for a google-scale disk-based database. *CoRR*, abs/2012.12501, 2020.

[2] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008.

[3] IBM Knowledge Center. Key compression. `https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.idad400/comp.htm`. Accessed: 2021-02-12.

[4] Surajit Chaudhuri, Usama M. Fayyad, and Jeff Bernhardt. Scalable classification over SQL databases. In *ICDE*, pages 470–479. IEEE Computer Society, 1999.

[5] Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[6] Yifan Dai, Yien Xu, Aishwarya Ganesan, Ramnatthan Alagappan, Brian Kroth, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. From wisckey to bourbon: A learned index for log-structured merge trees. In *OSDI*, pages 155–171. USENIX Association, 2020.

[7] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. ALEX: an updatable adaptive learned index. In *SIGMOD Conference*, pages 969–984. ACM, 2020.

[8] Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3123–3132. PMLR, 2020.

[9] Andrej Karpathy. Software 2.0. `https://medium.com/@karpathy/software-2-0-a64152b37c35`, 2017. Accessed: 2021-02-12.

[10] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD Conference*, pages 489–504. ACM, 2018.

[11] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. *CoRR*, abs/1808.03196, 2018.

[12] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking learned indexes. *Proc. VLDB Endow.*, 14(1):1–13, 2021.

[13] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Learning to steer query optimizers. *CoRR*, abs/2004.03814, 2020.

[14] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, 2019.

A Data Quality-Driven View of MLOps

Cedric Renggli[†] Luka Rimanic[†] Nezihe Merve Gürel[†] 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

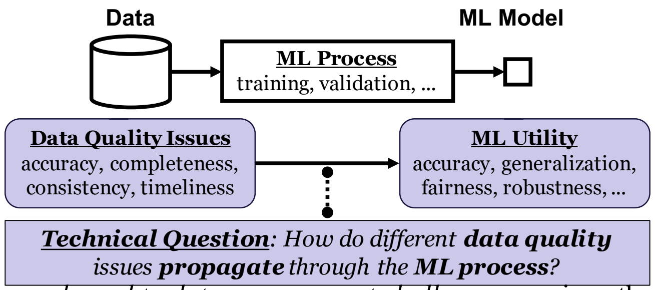
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

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

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.

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.


Data
ML Model
ML Process
training, validation, ...
Data Quality Issues
accuracy, completeness, consistency, timeliness
ML Utility
accuracy, generalization, fairness, robustness, ...
Technical Question: How do different data quality issues propagate through the ML process?

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 $h : \mathcal{X} \rightarrow \mathcal{Y}$ 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 $p(Y|X)$,

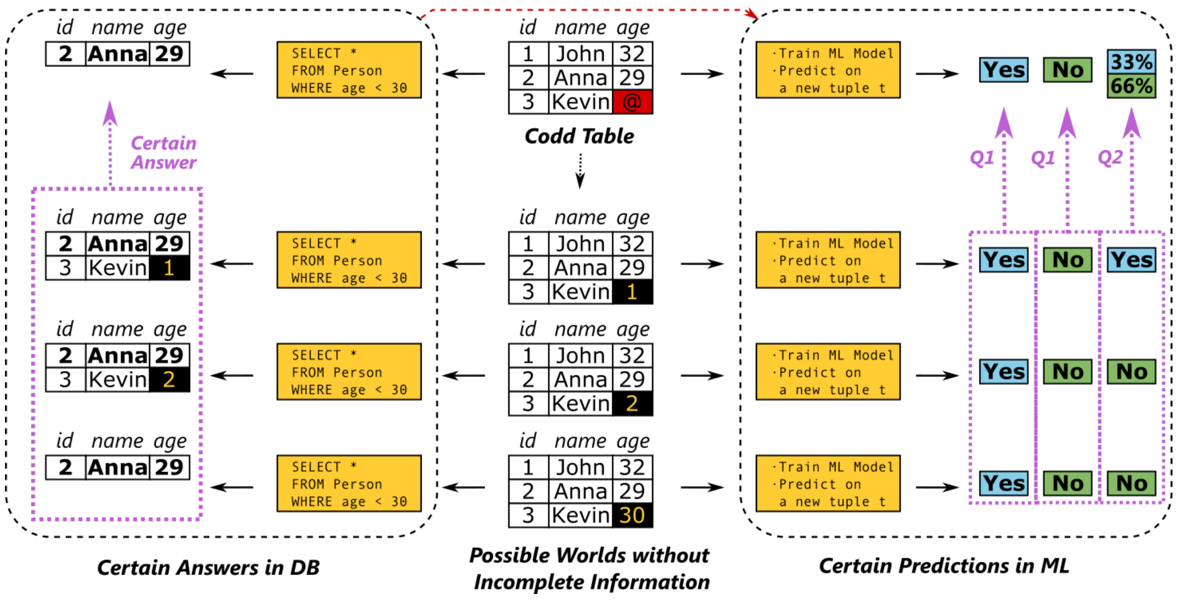

id name age
2 Anna 29
Certain Answer
SELECT * FROM Person WHERE age < 30
1 John 32
3 Kevin @
Codd Table
3 Kevin 1
3 Kevin 2
3 Kevin 30
·Train ML Model ·Predict on a new tuple t
Yes
No
33%
66%
Q1
Q2
Certain Answers in DB
Possible Worlds without Incomplete Information
Certain Predictions in ML

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*.

this is known as a *real drift* or a *model drift*, whereas when $p(Y|X)$ stays intact it is a *virtual 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 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,*

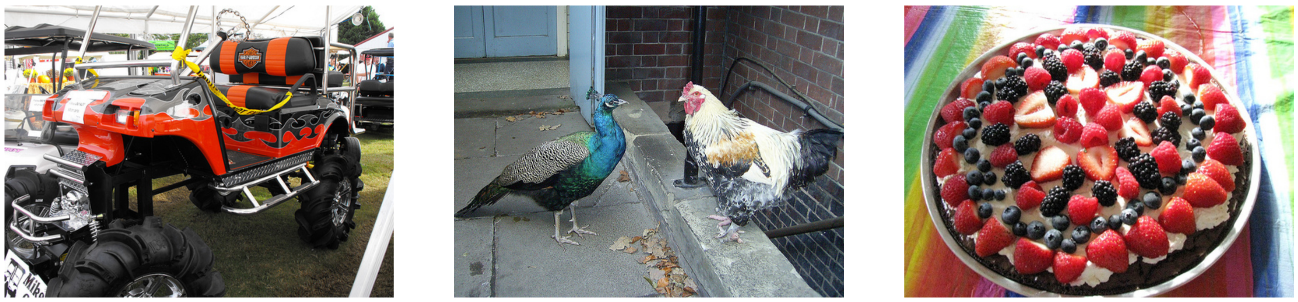
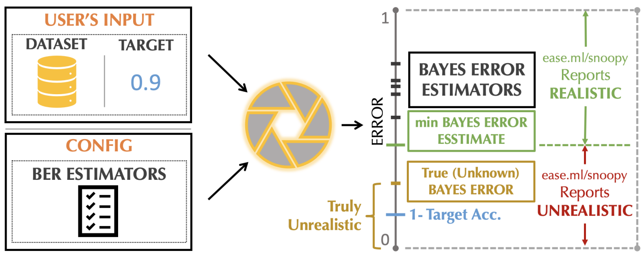


Figure 2: ImageNet examples from the validation set illustrating possible reasons for a non-zero Bayes Error. Image (#6874) on the left illustrates a non-unique label probability, image (#4463) in the middle shows multiple classes for a fixed sample, and image (#32040) on the right is mislabeled as a "pizza".

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, the image on the left 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. The center image 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. The image on the left 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

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

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.

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.

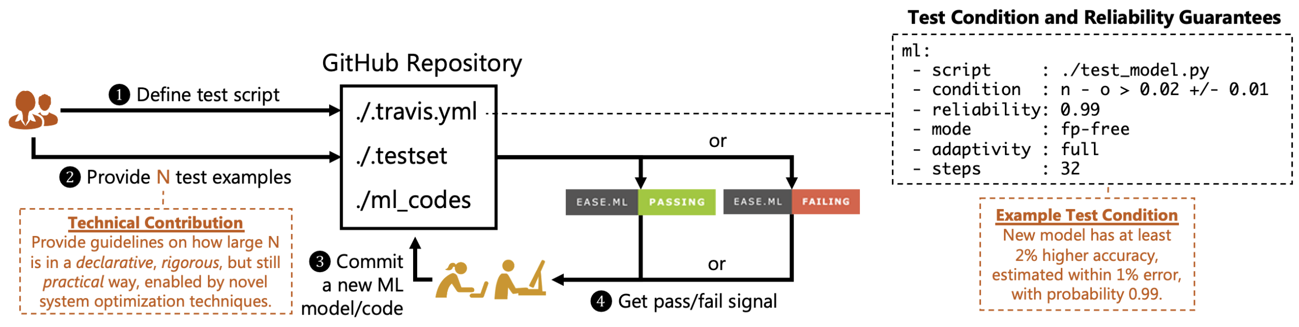
5 MLOps Task 3: Rigorous Model Testing Against Overfitting

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.


GitHub Repository
❶ Define test script
❷ Provide N test examples
./.travis.yml
./.testset
./ml_codes
Technical Contribution
Provide guidelines on how large N is in a *declarative*, *rigorous*, but still *practical* way, enabled by novel system optimization techniques.
❸ Commit a new ML model/code
or
EASE.ML PASSING
EASE.ML FAILING
or
❹ Get pass/fail signal
Test Condition and Reliability Guarantees
Example Test Condition
New model has at least 2% higher accuracy, estimated within 1% error, with probability 0.99.

```
ml:
 - script      : ./test_model.py
 - condition   : n - o > 0.02 +/- 0.01
 - reliability: 0.99
 - mode        : fp-free
 - adaptivity : full
 - steps       : 32
```

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

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`), 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.

From Cleaning before ML to Cleaning for ML

Felix Neutatz[1], Binger Chen[1], Ziawasch Abedjan[2], Eugene Wu[3]
[1]TU Berlin, [2]Leibniz Universität Hannover, [2]L3S Research Center, [3]DSI, [3]Columbia University
{f.neutatz,chen}@tu-berlin.de, abedjan@dbs.uni-hannover.de, ewu@cs.columbia.edu

Abstract

Data cleaning is widely regarded as a critical piece of machine learning (ML) applications, as data errors can corrupt models in ways that cause the application to operate incorrectly, unfairly, or dangerously. Traditional data cleaning focuses on quality issues of a dataset in isolation of the application using the data—Cleaning Before ML—which can be inefficient and, counterintuitively, degrade the application further. While recent cleaning approaches take into account signals from the ML model, such as the model accuracy, they are still local to a specific model, and do not take into account the entire application's semantics and user goals. What is needed is an end-to-end application-driven approach towards Cleaning For ML, that can leverage signals throughout the entire ML application to optimize the cleaning for application goals and to reduce manual cleaning efforts. This paper briefly reviews recent progress in Cleaning For ML, presents our vision of a holistic cleaning framework, and outlines new challenges that arise when data cleaning meets ML applications.

1 Introduction

Machine learning (ML) has gained widespread adoption in real-world problems that span business [1], manufacturing [2], healthcare [3], agriculture [4], and more. ML relies on - and is "programmed" by - training data. Thus, the quality of the training data is a fundamental ingredient toward robust and accurate models, and ultimately toward useful and reliable ML-based applications [5, 6, 7]. For this reason, data and ML engineers spend a tremendous amount of time—80% or more of a data scientist's time [8, 9, 10]—on wrangling and cleaning the required datasets for their ML applications.

Traditional data cleaning seeks to directly address data quality issues in a specific dataset. Given a structured dataset that potentially contains errors, it seeks to identify and/or repair those errors to derive a cleaned dataset that can be shared with the rest of the organization, or used by subsequent queries and applications without worry. Since cleaning occurs prior to, and often independent of the application, these techniques typically rely on error models to detecting duplicates or outliers, external constraint information (e.g., functional dependencies or integrity constraints), or human assessment and input (e.g., to recommend repairs or cleaning examples).

The separation of data cleaning and the application is not optimal. For one, it is hard for users to define, or even assess, the correct integrity constraints for the application. It is also hard to anticipate the different ways that the cleaned dataset will later be used. Further, improving a dataset could in fact *degrade* the application [11]. Thus, it is often unclear how a given cleaning intervention will affect the downstream application. For instance, is setting an outlier value to the median the best choice for a visualization dashboard, and does it even matter?

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Flights

Training Data	Test Data: Clean	Test Data: Dirty
Clean	0.33 ± 0.20	0.30 ± 0.17
Dirty	0.15 ± 0.17	0.10 ± 0.11

US Census

Training Data	Test Data: Clean	Test Data: Dirty
Clean	0.65 ± 0.05	0.19 ± 0.21
Dirty	0.16 ± 0.18	0.66 ± 0.04

Table 2: Model accuracy on Flights and Census with combinations of cleaned/dirty $\times$ training/test data.

Although these are not new issues in data cleaning, ML applications exacerbate these issues and introduce new and novel cleaning challenges. Table 2 reports an illustrative experiment where we train a classifier using AutoSklearn [12] and report its accuracy on a separate test dataset. We manually created clean and dirty versions of the FAA Flights delay [13] and U.S. Census [5] datasets, and split each dataset using 10-fold cross-validation into training and test sets. We find that whether or not cleaning is beneficial heavily depends on the application. For the Flights dataset, cleaned training data improves the model accuracy on both clean and dirty test data. However, cleaning the Census training data actually degrades the model accuracy on dirty data. In fact, training and testing on dirty data is as accurate as training and testing on clean data, yet requires no effort.

This experiment provides evidence that cleaning is not a local "one-and-done" process. In fact, **the appropriate cleaning intervention is dependent on the type of error as well as the rest of the application, and should be approached from this perspective**. Consequently, all of the complexities inherent in modern ML applications become complexities that affect how data is cleaned. In this paper, we argue that data cleaning needs to take an *end-to-end application-driven approach* that integrates cleaning throughout the ML application.

1.1 Data Cleaning in ML Applications

ML applications can be modeled as complex workflows that span an entire organization's data management, from data ingest to publishing data products to end users. Consider an e-commerce company that identifies users for promotional discounts (Figure 1). The data management team ingests user profile data from a third-party data source, and combines it with internal customer purchasing and browsing histories. The data is prepared by canonicalizing the user ids, extracting product information from the browsed pages, and ensuring that the expected attributes appear in each data record. Separate data science teams develop two models: the first estimates the likelihood that a given user will leave the service (churn), and the second estimates a user's preference for different products (affinity). These models are used in the promotion application to decide whether or not to show a promotion to an end-user (user in color), as well as the marketing team's internal dashboard and alert system that monitors the churn rate over time. Different teams (colored boxes) manage different parts of the workflow, and different engineers may monitor intermediate data at different points in the workflow (dashed lines and gray users).

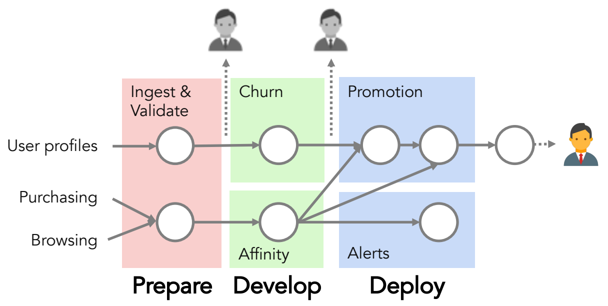


Figure 1: Typical workflow for ML applications

1.1.1 Cleaning within individual phases

The e-commerce application highlights the three prototypical phases in an ML application [14, 6]: the **Prepare** phase focuses on data ingestion and preparation; the **Develop** phase focuses on ML model development, training, and evaluation; and the **Deploy** phase uses the models in the production-ready application. Each high-level phase involves complex, multi-step data transformations and processing, and the steps are stitched together in a large workflow that often spans multiple teams within an organization. Each phase, as well as the ML application as a whole, poses novel challenges for data cleaning.

The **Prepare** phase typically focuses on restructuring the input data so that they conform to the expected "syntax" of the models. This can involve data extraction and wrangling from text documents [15], image resizing and reshaping [16], attribute type and domain checks [17, 6], as well as simple constraints that any downstream application will expect [18]. There has also been a large literature on cleaning approaches that seek to address "semantic errors" [19, 20, 21, 22, 23, 24, 25, 26, 13]. However, most approaches do not take the subsequent workflow into account, which the above experiment showed can lead to wasted effort or even worse application outcomes.

The **Develop** phase focuses on model development, training, and evaluation. Although the ML community has developed a multitude of robust model designs and training techniques [27], it is often better to directly address errors and biases in the data [5]. Cleaning in this phase can leverage the ML models and validation data to direct cleaning efforts to most improve model quality. In contrast to well-defined constraints that are available during the Prepare phase, quality for ML models is more challenging to define. Using test accuracy alone is susceptible to overfitting, and other factors such as model generalization must be taken into account but are hard to quantify. Further, ML models are often trained on nonrelational data, such as images, natural language, and unstructured documents, which require new types of cleaning interventions.

In the **Deploy** phase, trained models go into production to directly serve recommendations to end users, or as part of data analytics that power e.g., dashboards or monitoring systems. Rather than make cleaning decisions based on model quality, cleaning in this phase can leverage the ML application's user-facing results to better identify erroneous input data and save the effort of cleaning data that does not affect the output. However, cleaning data is harder as it may not be clear what upstream dataset (and which records) to clean, how intermediate transformations in the workflow might affect possible cleaning interventions, nor how to translate error signals into actionable interventions. Further, external signals in this phase may override or contradict cleaning decisions made earlier in the workflow.

1.1.2 Cleaning in ML Applications

Stepping back, the goal of data cleaning is ultimately to improve the ML application. This perspective presents three main challenges. First, the notion of data quality is more varied and often unclear. Even in traditional data cleaning, each phase has different goals: data preparation enforces syntactic constraints, model development seeks to improve model quality, while deployment seeks to improve the ML application. Quality will often rely on a user's qualitative judgments based on application-level outputs. For instance, an analyst that examines a dashboard is well positioned to spot anomalies in the visualizations. On the other hand, human judgments may not always be trustworthy nor correct, and attention is a limited resource.

Second, data management, cleaning, and error detection is fragmented across different roles and different parts of the workflow. For instance, a data management team may be responsible for data ingestion and preparation, data scientists design and train models, ML engineers then productionize and publish models that can be deployed by application developers. At the same time, the appropriate cleaning intervention (if any is even needed) depends on the nature of the error *and* the model [28, 29, 30] (and application). Thus, domain expertise is localized, visibility into the full workflow is limited, yet their cleaning decisions are conducted and felt globally.

Third, ML data is both complex and dynamic. Relational, multimedia, and natural language data all require different types of interventions. ML models are sensitive to population-level errors that can bias the model. Further, different datasets play different roles throughout the workflow. Ingested data is preprocessed into training data, which is distinct from test and inference data, and they are both distinct from data used for general analytics. Finally, data constantly changes over time. For instance, inference data today may become test or training data tomorrow. The population-level dataset statistics, data schemas, and types of errors can change over time.

Ultimately, data quality management in ML applications requires the ability to make informed cleaning decisions at different parts of the ML application, for different types of data and errors, and for different purposes as the data, models, and workflow evolves. This requires coordination across the workflow so that cleaning decisions can account for downstream needs, while downstream steps can help inform upstream cleaning.

1.2 Looking Forward

To this end, we propose an *end-to-end application-driven approach toward cleaning*. Specifically, we envision a holistic framework that connects involved stakeholders of the entire ML application's workflow. This framework manages the provenance of cleaning operations throughout the ML application. The provenance enables stakeholders to provide signals, cleaning suggestions, and expertise at any part of the workflow. Then, this feedback can be leveraged for cleaning in the rest of the workflow. The framework should also

- Make it possible to change/retract cleaning decisions of early phases based on feedback obtained in later stages of the workflow;
- Expose an abstract view of cleaning operations so that feedback on data quality and prior cleaning operations will be usable across all phases. Feedback on data quality will include business rules, human annotations, and the application's internal and intrinsic scores;
- Enable the extension and variation of the cleanliness definition in accordance with the use case at hand. It will allow for diverse error types, the weighting of errors, and error types based on importance.

Paper Scope: There are two forms of data cleaning: repairing errors at the instance level (e.g., find and fix erroneous records or values), and to fix population-level errors that can arise from distributional shift, biased sampling, or record errors. This paper primarily focuses on the former notion, and we briefly discuss population-level errors in Section 4.
In the rest of our paper, we first review the state-of-the-art in data cleaning (Section 2) and identify potential building blocks of our vision. Then, we layout our holistic cleaning approach and its desired component in detail. Finally, we shed light on how this vision would also address new challenges that arise in ML applications.

2 Data Cleaning and ML

This section introduces and distinguishes traditional data cleaning techniques from those designed for ML. Specifically, *cleaning before ML* approaches perform data cleaning independently of the downstream ML applications, whereas *cleaning for ML* approaches explicitly dependent on them.

2.1 Cleaning before ML

Traditional data cleaning was performed during data ingestion as part of an Extract-Transform-Load (ETL) workflow. Administrators perform data wrangling [15, 31] and transformation [32] so that the data conforms syntactically (e.g., the appropriate schema, data types) to the application requirements. They further specify constraints (e.g., domain constraints, functional dependency, distribution) for all database instances. The database system enforces these constraints and possibly repairs the database to address any violations.

ML applications often combine a wide diversity of datasets, and it can be challenging for an administrator to define constraints and repair logic manually. Thus, a number of data cleaning techniques use data-driven approaches to help detect as well as repair datasets. The following are termed *Cleaning Before ML* techniques because they are primarily applied to a training dataset before model development and training, although some techniques may also be used to clean test data during model inference. These techniques make strong assumptions about the impact of their repairs on the downstream model and application performance.

Non-learning Approaches. Data cleaning systems for error detection and/or error repair are non-learning if they directly rely on user-provided rules or specifications (which we generically term *cleaning signals*). For instance, pattern-based systems [33, 15] leverage user-specified regular expressions or patterns to identify syntactic text errors, while rule-based systems [20, 34] rely on user-specified functional dependencies to identify values that violate the constraints. Outlier detection systems [35, 36] are tuned to specific data distributions and thresholds to identify outlier values. Finally, systems may leverage external data such as a knowledge base [19] to validate the instances of a dataset. For instance, if the knowledge base contains address data, KATARA can leverage it to identify mismappings between cities and ZIP codes of a given dataset.

In the context of ML applications, cleaning systems [17, 6] help developers validate that a training set exhibits expected properties. For instance, Deequ [17] lets users declaratively write constraints, such as based on uniqueness, completeness, and value distribution, and compiles them into "data unit-tests" that can be executed at scale. Many production ML platforms [6, 37, 38] similarly validate that training data contains expected attributes in its schema, and that data distributions have not drifted beyond a threshold.

Error repair is traditionally posed as a constraint satisfaction problem. Given a set of data constraints, identify the minimum number of interventions so that there are no constraint violations in the dataset. Variations of this minimality principle may include additional objectives (e.g., statistical distortion [39]), or allow a small number of violations. The above approaches lean heavily on the user or administrator to provide accurate and useful cleaning signals, which may not always be feasible. In addition, each approach is designed for a specific type of error, and thus exhibits low recall in practice. ML applications may need to combine many cleaning systems to address different types of errors, or to make use of different types of cleaning signals. We next see how learning-based cleaning seeks to address some of these limitations.

Learning-based Approaches. Learning-based cleaning leverages ML to identify and repair errors. These fall under two main directions. Ensembling approaches combine existing cleaning systems into an ensemble and use learning to decide which to use. The alternative is to treat detection and repair as ML problems that respectively predict whether a record is erroneous, and what the correct value for an erroneous attribute value should be.

- **Error Detection:** Metadata-Driven Error Detection [24] and Raha [25] are examples of the ensembling approach, while ED2 [26], DataWig [40, 41], HoloDetect [22], and Picket [42]. The latter approaches model each record or cell, along with any cleaning signals, as features used to classify it as erroneous or clean. Naturally, learning-based approaches rely on pairs of erroneous and clean records as training examples, and semi-supervised strategies avoid the need to manually provide examples.

  For instance, ED2 [26] uses active learning to acquire clean/erroneous labels for records that the model is uncertain about, while Raha [25] further clusters records by similarity and acquires labels on a per-cluster basis. HoloDetect [22] uses data augmentation by learning observed error patterns and applying the patterns to generate synthetic errors. Picket [42] does not rely on any external labels and instead uses self-supervision to learn an error detection model that can be applied during training or testing.

- **Error Repair:** Repair systems such as Baran [13] and HoloClean [21] combine different types of cleaning signals to more accurately repair errors. Baran [13] is an ensembling approach across a library of error repair strategies, and uses active learning to train a model that predicts which repair strategy to use. The library of strategies is extensible, and can include additional predictive models that propose fixes. Furthermore, it uses

Table 3: Taxonomy of recent cleaning for ML systems.

System	Intervention Type	Semantics	Signals	Model Type
ActiveClean	Manual record repair	Model	Model Gradient	Convex
CPClean	Manual value imputation	Model	Prediction Certainties	Nearest Neighbor
BoostClean	Cleaning program	Blackbox	Model Validation	Any
AlphaClean	Cleaning program	Blackbox	Custom Objective	Any
AutoSklearn	ML pipeline	Blackbox	Model Validation	Any
Rain	Record repairs	Relational Workflow	User Complaint	Differentiable

transfer learning and label propagation to reduce the required amount of user labels. HoloClean [21] combines integrity constraints, external data, and statistical profiles into a single factor graph model, and uses the model to predict the appropriate value for identified errors.

Stepping Back. Traditional data cleaning systems are designed around specific types of cleaning signals, whereas learning-based approaches use ensembles or models to combine a variety of signals and achieve higher recall. Weakly semi-supervised techniques such as active learning, label propagation, data augmentation, and transfer learning help reduce the degree of user involvement. However, these approaches clean a dataset in isolation. We next describe cleaning systems designed specifically in the context of ML applications.

2.2 Data Cleaning for ML

Data cleaning fundamentally relies on external cleaning signals (constraints, patterns, examples, etc.) to detect and repair data errors. In contrast to the explicitly user-specified signals described in the previous subsection, *cleaning for ML* approaches leverage the downstream model or application to define cleaning signals that incorporate higher-level semantics. This subsection describes five recent cleaning for ML systems developed in the data management community. Our rationale for this selection is that these approaches consider the semantics of the ML task and a subset of its emitted signals to deploy cleaning routines on the training dataset. Thus, we distinguish them along four dimensions (Table 3): intervention type, semantics, signals, and model type. The *intervention type* corresponds to whether the goal is to directly repair a dataset, or to generate a cleaning program that can be applied to future datasets as well. The *semantics* describe whether the approach relies on the downstream model or application semantics, or treats it as a black box. The *signals* categorize the type of cleaning signal(s) that are used. The *model type* describes which group of ML models is supported by the corresponding approach.

ActiveClean. ActiveClean [43] is a data cleaning system for models with convex loss functions. It treats a model trained on a dirty training set as a suboptimal point along the loss function over the (unknown) clean training set. It then treats cleaning as a stochastic gradient descent (SGD) problem, where in each step, the system samples and asks the user to clean records that are expected to shift the model along the steepest gradient. The use of convex models allows the system to derive convergence guarantees. ActiveClean simply chooses which training records to repair next, but relies on the user to perform the actual repair. Its cleaning signals combine the user's repairs and the model's loss function. Finally, it relies on the model's convexity to guarantee convergence.

CPClean. CPClean [44] incrementally cleans a training set until it is certain that no more repairs can possibly change the model predictions. To do so, it proposes the concept of Certain Predictions (CP) for nearest neighbor classifiers [44]. A test record is certainly predicted if all classifiers trained on all possible repairs of an incomplete training set would yield the same prediction. CPClean develops an efficient counting query to compute the number of possible models that support a specific classification of a given test point, and uses this to quantify the impact of repairing a given error or missing value. Thus, given a validation set, CPClean seeks to only repair the training records needed so that all validation records can be certainly predicted. CPClean uses the validation set and the counting query as the primary cleaning signals, but relies on the user to perform the actual repairs. It is specially designed for nearest neighbor models, and leverages their robustness to small training perturbations.

BoostClean and AlphaClean. BoostClean [45] treats cleaning as a boosting problem, and outputs a cleaning program that can be applied to training or test records. It is specialized to conditional value errors, where an error can be specified using a simple predicate. Given a predefined library of parameterized detection and cleaning functions, it automatically selects a sequence of cleaning operations that will maximize the model's accuracy on a validation set (or the training set). In each step, BoostClean uses statistical boosting to choose a pair of detection and cleaning functions and configures their parameters, and applies it to the training set to derive a new model. BoostClean diverges from prior cleaning systems in that users do not need to manually repair records. Instead, users simply specify appropriate detection and repair functions for the domain. BoostClean leverages the model's training or validation accuracy as the primary cleaning signal.

AlphaClean [46] is a similar system that generates a sequence of cleaning operations from a library of parameterized cleaning frameworks, including existing systems such as dBoost [47] and HoloClean [21]. In contrast to boosting, it uses parallelized tree search and learns pruning heuristics to reduce the search space. Since AlphaClean uses a generic search procedure, the user is free to define their own objective function as the cleaning signal (e.g., the model accuracy, the number of constraint violations, a combination of the two).

AutoSklearn. AutoSklearn [12] similarly uses the model validation accuracy as the primary cleaning signal to generate an end-to-end ML pipeline. The pipeline includes preprocessing operations, hyperparameter selection, and model training. To restrict the search space, the preprocessors are limited to value imputation based on mean, median, or most-frequent values, and are either applied to the whole dataset or not at all (e.g., no conditional repairs). Further, AutoSklearn focuses on repairs and does not perform error detection.

Rain. Rain [48] seeks to go beyond an individual model and considers the downstream workflow. It executes relational workflows consisting of relational operators and inferences made by a differentiable model. For instance, the query "count number of users that are likely to churn" uses a predicate that filters by predicted churn, which is estimated by a linear regression model. The user can annotate errors (termed "complaints") in the outputs or intermediate results. For instance, the user may specify that the total price in January should be 40 instead of 100, or all values in the output are too low. Rain then ranks erroneous records (and their repairs) in the training dataset based on how much it will resolve the complaints. To do so, Rain models the workflow as a differentiable function over the model prediction probabilities. The function can leverage the influence functions framework [49] to estimate the gradient of the query result with respect to epsilon changes to the training dataset—the removal of a training record or the addition of a new (cleaned) record. Their experiments show that a single annotation of an aggregate result can more effectively identify training errors than manually labeling hundreds of model mispredictions. By supporting complaints over workflow outputs, it empowers users to specify constraints within the context of the application's downstream semantics.

Cleaning in the ML Community. The ML community has long studied methods to address or circumvent data errors in learning. A common approach [36] uses a model to classify training records as erroneous—based on model uncertainty, voting, clustering, etc—but is primarily limited to deleting records or repairing labels. Although some errors provably require data cleaning [28], some errors can be good—training with noise is a form of regularization [50], and is the basis of data augmentation methods [51]. Robust estimation methods re-weigh, filter, and otherwise adapt the estimation procedure to be insensitive to outlier training data, including worst-case outliers [52, 53]. For instance, models that perform local averaging such as weighted and interpolated nearest neighbors are naturally robust to noisy training data [54]. Models may be certified as robust to adversarial test errors [55, 56, 57]. The above techniques are susceptible to systematic errors. Although they can be explicitly modeled [58], this requires considerable modeling expertise, and it may not be clear what errors are even present *to* model. To this end, open source libraries such as Dabl [59], Dataprep [60], and Facets [7] combine visualization and common detection methods to aid data preparation and cleaning.

Stepping Back. The database community has made significant contributions toward cleaning approaches that can be deployed in the Develop and Deploy phases. State-of-the-art cleaning for ML systems [43, 45, 44, 46, 44] leverage application-specific signals to guide human cleaning or to choose cleaning operations. Some of them are restricted to cleaning training data and cannot generalize to unseen data [43, 44]. Debugging systems, such as Rain, consider the output of an entire workflow to trace-back training instances that lead to wrong results. A recurrent problem is that each proposed solution only covers a specific aspect of cleaning, such as specific error types, dataset components, or ML models. As a result, a successful attempt at cleaning a dataset for deployment forces analysts and scientists to use multiple disconnected approaches.

3 Cleaning for ML Applications

The Introduction highlighted three major challenges toward data cleaning for ML applications. First, the definition of data quality and what a data error even means is not only user-defined, but is also model- and application-specific. This makes these definitions difficult to foresee and articulate during the model design and data ingestion phase. Second, domain knowledge and expertise is fragmented across the different users—developers, ML experts, IT staff, and end users—that each have limited visibility into the ML application's workflow (or workflow for short). Third, ML applications are complex, dynamic, and heterogeneous. They combine and process relational and non-relational data streams whose properties may continuously change over time.

As a step toward tackling these challenges, this section describes a research agenda that extends data cleaning to encompass the needs of ML applications. We first center the agenda around a high-level architecture, and then highlight a number of promising research challenges in each of the architectural components.

3.1 Architecture

Data cleaning is fundamentally human-centered. Humans design the ML application with specific goals in mind; humans build, operate, and maintain each component of the application's workflow; and humans use the resulting application to accomplish their tasks. Thus, it is crucial to "bring cleaning to the user." At any point in the workflow, it should be easy to inspect the data that flows through, and provide data quality signals that help to make error detection and repair decisions.

We envision data cleaning as an instrumentation layer that sits atop the entire application workflow (Figure 2). Its purpose is to use elicit cleaning signals throughout the workflow to decide which data to clean, how to improve data quality, and where in the workflow to instrument which cleaning operations. The dashed lines denote workflow steps that serve as inspection points for users and provide cleaning signals to the *Signal Collector*. The inspected data can first be processed and transformed (e.g., by a script or query) before being rendered to the user (e.g., as a table, interactive visualization). The key characteristic of any data interface is that it can be annotated with data errors that the user sees, expected trends or data properties, or any other signal that can aid the cleaning system. These signals are collected and routed to *Cleaning Optimizers* that make local or global cleaning decisions about what cleaning operations to install or test. The *Instrumenter* then augments the workflow with cleaning operations (red squares), and monitors their effectiveness over time. The monitoring signals serve as additional feedback to improve the optimizers. The user input-optimize-clean loop is a classic approach toward user-centered data cleaning [62]. The key distinction of the proposed design is that this loop permeates throughout the ML

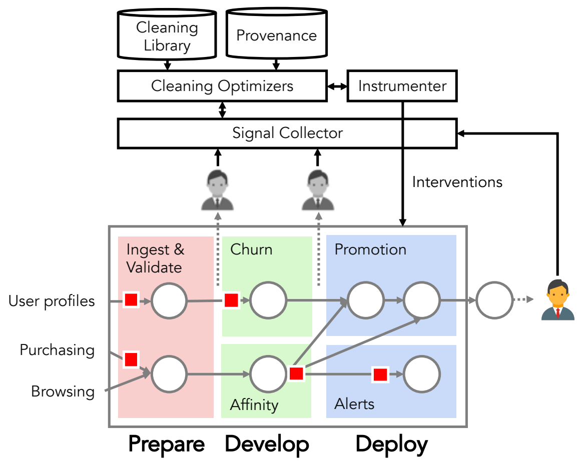


Figure 2: Envisioned Holistic Cleaning Framework.

application. Data that is deeper in the workflow encodes increasingly more application semantics and qualitatively changes the scope and type of cleaning signals that users can provide. For instance, signals collected during the Develop phase can leverage model-specific statistics to quantify data quality, while signals in the Deploy phase, such as data presented to the end user, encode utility toward the user's task. This parallels visual analytics, which transforms, filters, and summarizes data as visualizations to surface unanticipated trends and patterns that are not visible, nor easily expressible, in the raw data.

3.2 Signal Collection

Cleaning signals are external information that initiate and inform data cleaning in the workflow. Existing cleaning approaches assume a single cleaning signal is collected at a specific point in the workflow, and is coupled with the cleaning algorithm (Section 2.2). In contrast, we envision a framework that can request and collect cleaning signals from any point in the workflow at varying levels of applicaton abstractions.

3.2.1 Signal Types

In general, a cleaning signal is any information that can aid data cleaning. Signals can be provided by the users (e.g., through a visualization), derived from the application (e.g., model accuracy), or programmatically generated (e.g., an alert, or constraint violation). We initially propose to support the four types of signals—constraints, rules, annotations, and objectives—commonly used in existing cleaning systems.

Integrity constraints [20], such as functional dependencies, denial constraints, can be specified for any input, intermediate, or output dataset in the workflow. They can be used directly, or be used to trigger alert signals when they detect violations. Similarly, ML-specific constraints may specify e.g., fairness [63], robustness, or generalization properties. Constraints may be user-specified or automatically generated.

Rules include predicates to remove records or ETL transformations, and may trigger signals when they fail. Manual annotations of a dataset specify erroneous records or values. For instance, a user may annotate that a value should be 50 instead of 30, or provide a label for a training record. Annotations can also be automatically computed in the workflow through application-specific scores, such as the model validation score [45, 12, 46], prediction certainty [44], and gradient information [43]. Finally, the cleaning system can directly listen on the actual application objective, such as classification accuracy.

3.2.2 Eliciting Signals

To elicit the various user and application-specific signals from the ML workflow, it must be possible for the cleaning system to inspect workflow data. This leads to two main questions:

Where to elicit? When used as part of data debugging or engineering, there are natural points in the workflow where users are accustomed to provide signals. These typically include data ingestion, before and after model training, and the application output. However, there are likely other useful points to assess data quality to detect potential errors, or to elicit signals to aid later cleaning. For instance, it is often difficult to coordinate across organizations, thus collecting signals at steps that cross organizational boundaries may be prudent.

A major research question is what signals should be elicited during data cleaning. Once a user has reported a data error (say, in the application output), where should additional signals be collected that can improve the cleaning algorithms? For instance, the algorithm may generate candidate repairs and ask the user to label them; or ask users to annotate other application outputs to choose between possible repairs; or ask for integrity constraints to narrow a search space. This is also an opportunity to automatically collect signals (e.g., model accuracies) without any user effort.

What elicitation interfaces? Users should be able to use interfaces to interactively e.g., confirm or highlight errors in the data [26, 25, 13], provide example records [13, 32], and specify constraints or rules [20]. Assuming that workflow data is structured, an initial approach is to provide a library of general data presentation options, such as automatic data visualization [64, 65], spreadsheet interfaces, or a scripting environment. Further, systems can interactively explain error causes and how they are detected/corrected [66, 67, 68, 69, 70, 71]. Here, one research opportunity is to design recommender systems that propose the best setup from a large number of potential visualization methods based on previous user interaction and metadata. Collecting signals at these various phases enables to clean training, test, and deployment data with the same cleaning goals in mind to avoid distributional gaps.

3.3 Cleaning Optimizers

Signals represent the application or user's expectations about data quality, and can be both used to initiate data cleaning or improve cleaning quality. For instance, a user annotation may notice an output error, which triggers the need to cleaning a training dataset; an integrity constraint on the training data helps prune the algorithm's search space since it is already enforced. The purpose of the cleaning optimizer is to combine different signals and propose a pipeline of cleaning operations that detect and repair errors in a dataset, and where to apply them. Each cleaning optimizer may be an existing signal-specific cleaning system (such as those in Section 2) or new "holistic" optimizers that combine disparate signals. Each optimizer should specify the subset of the workflow it can be responsible for (e.g., model training, data preparation, relational sub-workflows), the signal types it requests, and crucially, the types of interventions it accounts for. We distinguish optimizers along two dimensions: whether the optimizer is local to a phase or operator in the workflow, or takes a subset or whole workflow into account; and whether the optimizer treats the workflow as a white or black box.

3.3.1 A "Holistic" Cleaning Optimizer

An open question is to what extent different signals throughout the workflow can be integrated to holistically clean data in the workflow. The different signal types naturally express hard constraints, soft constraints, and objective functions, thus may be amenable toward translating into a singular optimization problem. Prior works such as HoloClean [21] and Baran [13] have shown how this is possible using factor graphs and ensembles for specific combinations of signals. Many existing works focus on specific classes of data errors, signals, and workflows, and a major challenge is how to support a wider range of signal types, interventions, different collection points, and incorporating the workflow semantics.

3.3.2 Workflow-independent vs -dependent Optimizers

A workflow-independent optimizer is specific to a single workflow operator and does not use any workflow semantics. Most existing systems that clean [21, 20], wrangle [15, 20, 33], or transform [32] a specific dataset are workflow-independent. In contrast, a workflow-dependent optimizer cleans data at a given point in the workflow by using signals (e.g., model accuracy, user annotations) from downstream and/or upstream. Although recent cleaning systems incorporate downstream signals [45, 46, 44, 12], it is an open question how to integrate multiple cleaning signals collected from different points in the workflow and optimize in a workflow-dependent manner.

3.3.3 White vs Black Box

Workflow-dependent optimizers combine workflow semantics and cleaning signals. For instance, if a user specifies errors in the application output, and the system wishes to clean ingested data during the Prepare phase, it needs to consider all of the workflow's operator semantics to assess how different cleaning decisions would affect the output. Optimizers can be either implemented as a white or black-box approach.

In white-box approaches, the optimizer has access to signals of every single transformation in the ML pipeline, and leverages well-defined workflow semantics (e.g., in relational workflows) to encode the pipeline logic into constraints and/or an optimization objective. For instance, Rain encodes the SQL query as a relaxed provenance polynomial, and can evaluate custom record-level interventions. A white-box-specific optimization is to reduce the overhead of retraining by avoiding repetitive computation through partial retraining [72, 73] or caching [74, 75].

In contrast, black-box approaches ignore intermediate results and simply reason about the workflow input and output. They may translate cleaning signals into optimization constraints and objectives, and use generic black-box optimization techniques such as meta-learning techniques or Bayesian optimization to search the space of cleaning interventions. Existing meta-learning approaches that ensemble many cleaning algorithms (e.g., Raha [25], Baran [13]) only encode dataset properties in their feature vector and ignore the semantics of the application workflow. BOExplain [76] treats workflow cleaning as a blackbox hyperparameter search problem. Ultimately, the key challenge is to define an effective but concise search space, and to create a workflow representation that facilitates meta learning for ML-dependent cleaning.

White-box approaches can be more accurate and efficient, but do not readily support custom logic or user-defined functions. On the other hand, black-box approaches can be applied to any ML workflow, but can be very inefficient. A potential middle-ground is to approximate the workflow using heuristics, or making assumptions about the workflow operator semantics. For instance, the system might approximate an operator using an ML model [44], or symbolically execute the operator to derive a logical expression [77, 78]. Another interesting direction is to trace back the error causes to the corresponding workflow components [66].

3.4 Scope Refinement

When there are many data sources, datasets, and annotations, a fundamentally challenging problem is knowing which datasets are the candidates that may be responsible for the annotations. We call this the *Scope* of the annotations. There are several promising directions for scope refinement.

One approach to identify the datasets to clean is to assess their potential impact on the error annotations, either directly using sensitivity analysis [48], or using a framework akin to certain predictions [44]. Assuming that the dataset to clean is known, one way to refine scope is to use feature [79] or instance selection [80] to identify the most important columns and rows in the training data. ActiveClean and CPClean follow the instance selection approach [43, 44]. A third direction is to narrow the scope to the provenance of the annotated data [48]. However, this can still be a large set of records. For instance, the end-user-facing output of the ML applications trivially depends on all of the datasets (and a subset of their records).

Research opportunities in this field are to combine workflow-dependent optimization with scope refining strategies. The optimizer could generate different cleaning pipelines for different scopes of the data. For example, it could generate a set of cleaning operations based on functional dependencies for categorical data and use a Gaussian outlier detection technique and use a “replace with median value” as the default cleaning operation for numerical features. Some cleaning pipelines might be more general and cover a larger scope of the dataset leading to unnecessary cleaning efforts but instrumenting individual pipelines, which are simpler, for specific subsets of the dataset might introduce an unnecessary overhead. Automatically identifying the best scope refinement and the best trade-off for a dataset at hand is an interesting research problem. Being able to distinguish different scopes of a dataset also enables to prioritize cleaning efforts for the most significant scopes. Thus, one can decide to instrument cleaning on dataset subsets that are small but significant to speed-up real-time model and configuration testing in the Develop phase.

3.5 Instrumenter

Once the cleaning optimizers have chosen promising interventions, they need to be instrumented into the workflow. Although most systems support dynamic instrumentation, it alone is not sufficient, as the workflow and the data properties may differ from the assumptions made during optimization, and can also change over time. In real-world applications, data constantly changes over time [81, 34, 82]. The challenge is to identify when the distribution shifted too much so that any of the installed cleaning rules do not apply anymore. Thus, it is crucial to provide assessment and monitoring functionalities to aid the engineers to manage the data cleaning process.

One can investigate how to backtest or speculatively test interventions before installing them. Another interesting direction is to define an algorithm that periodically or continuously assesses the effectiveness of past cleaning interventions. Ultimately, the monitoring approach helps the data cleaning "team" to manage the disparate cleaning signals collected throughout the workflow to understand which are reliable, are consistent with each other, and which are used for a given intervention. A practical challenge here is to efficiently keep track of instrumented cleaning operations and data changes. Existing approaches that ensemble cleaning operators have to keep track of thousands of operators [25, 13] on each dataset. One could think of index-based solutions to scale the trace-back requests for large datasets or develop models that learn and predict the relationships.

4 New Challenges of Cleaning for ML

The previous section outlined our vision of holistic and flexible data cleaning for ML applications, and research opportunities related to the different components.

Data cleaning systems, including the proposed system in this paper, have historically been designed for structured, e.g., relational, and persistent datasets. Within this scope, they have focused on record and value-level errors, and data quality metrics are designed for these errors in mind, e.g., number of constraint violations, outlier-ness of a value. However, there are a number of challenges that go beyond the scope of our proposed vision. The type of input data used in ML workflows goes beyond structured data and can become less tangible through entanglement of models.

The notion of data quality is also much broader in scope for ML applications, and includes population-level errors, as well as social and cultural norms. This subsection describes these novel challenges and provides pointers for further consideration.

Unstructured data. ML approaches are especially successful for unstructured data, such as images, sound, and video. Traditional data management cleaning approaches and their cleaning operations focus on numeric, categorical, and textual data [26, 25, 13, 21, 22]. The proliferation of generative models offers the potential to similarly repair unstructured data by transforming e.g., images in semantically meaningful ways.

Model entanglement. This phenomenon arises when a model (or a component in general) compensates for errors in an upstream model [11, 83]. In this setting, the models are entangled, and improving any model individually will degrade the end-to-end performance. In general, hidden dependencies across different parts of the application can severely complicate data cleaning, and necessitates the need to incorporate downstream signals and workflow logic into the cleaning process. Identifying and disassociating entangled components is another important problem to tackle.

Population-level Errors: This paper focused on record-level errors, however population-level errors also impact ML applications today. These may be due to biased sampling, or systematic errors from data generating processes or preprocessing logic, and manifest as biased models and biased predictions. In these contexts, even cleaning all of the records in a dataset may not fully address the distributional biases. These errors can affect the ML application in not just biased predictions—differences between training and test distributions, or distribution drift over time can all introduce application-level issues.

Although there is work in detecting distributional shifts in a dataset [84, 85], accounting for group-wise errors (and interventions) the downstream implications is considerably more challenging than doing so for an individual record due to interaction effects [86]. Furthermore, continuing the research on pre-processing techniques [63] to avoid bias and inspecting pipelines for violating data patterns [87] is a promising direction to explore.

Social/cultural norms: Another new generation of errors are violations against social and cultural norms. For instance, one should avoid training on text, image, sound, or video corpora that contain hate speech [7], misinformation [88], or privacy violations [89, 90]. While some of these issues could be identified with lists of forbidden words, many might not be initially obvious to a human. The problems might appear in the downstream application in a more tangible form, which motivates to design algorithms that can trace the results back to data. These re-emerging quality dimensions are also changing over time. A potential direction is to design active-learning systems that support users to continuously define these norms.

Robustness Prediction: An increasingly important field in ML is safety [42]. Out-of-distribution [91] or adversarial test records can cause the ML model to make wrong predictions. ML-driven cleaning has the potential to help tackle these issues. Although individual value errors may be hard to detect, identifying erroneous records with respect to the model or application may be possible.

Clever Hans. The ML pipeline is expected to work well (to generalize) on new unseen data—training data used to update models, data used to make predictions, or any other data used in the application. Guiding the cleaning efforts with signals, such as validation score or uncertainty, might mislead the cleaning efforts through spurious correlations (Clever Hans phenomenon [92]). While in the Develop phase the cleaning might lead to higher accuracy, because it used the spurious correlations, we might end up with lower performance because these correlations are not prevalent. E.g., Lapuschkin et al. [92] found that their trained ML model used the source tag *horse_photo_archive.de* to classify images as horses instead of actual horse characteristics, such as its tail or its head. While the model performed very well on that dataset, it clearly missed the goal of capturing the visual features of a horse. Cleaning that is guided by accuracy might aggravate this problem. To prevent Clever Hans phenomena, it is not enough to have a feedback loop from the Deploy phase. A promising direction is to make model explanation an integral part of data cleaning.

Acknowledgments

This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A); Eugene Wu was funded by National Science Foundation awards 1564049, 1845638, and 2008295, Amazon and Google research awards, and a Columbia SIRS award.

References

[1] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, “Deep direct reinforcement learning for financial signal representation and trading,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 3, pp. 653–664, 2017.

[2] P. Priore, D. de la Fuente, A. Gómez, and J. Puente, “A review of machine learning in dynamic scheduling of flexible manufacturing systems,” *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 15, no. 3, pp. 251–263, 2001.

[3] J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, B. Haghgoo, R. L. Ball, K. S. Shpanskaya, J. Seekins, D. A. Mong, S. S. Halabi, J. K. Sandberg, R. Jones, D. B. Larson, C. P. Langlotz, B. N. Patel, M. P. Lungren, and A. Y. Ng, “CheXpert: A large chest radiograph dataset with uncertainty labels and expert comparison,” in *AAAI*, 2019, pp. 590–597.

[4] K. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine learning in agriculture: A review," *Sensors*, vol. 18, no. 8, p. 2674, 2018.

[5] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: a study for evaluating the impact of data cleaning on ml classification tasks," in *ICDE*, 2021.

[6] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *SIGKDD*, 2017, pp. 1387–1395.

[7] P. Lee. (2016) Learning from Tays introduction. [Online]. Available: https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/

[8] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang, "The data civilizer system," in *CIDR*, 2017.

[9] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Godwal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, J. Leeka, K. Park, H. Patel, O. Poppe, F. Psallidas, R. Ramakrishnan, A. Roy, K. Saur, R. Sen, M. Weimer, T. Wright, and Y. Zhu, "Cloudy with high chance of DBMS: a 10-year prediction for enterprise-grade ML," in *CIDR*, 2020.

[10] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *TVCG*, vol. 18, pp. 2917–2926, 2012.

[11] S. Amershi, A. Begel, C. Bird, R. DeLine, H. C. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: a case study," in *ICSE*, 2019, pp. 291–300.

[12] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *NeurIPS*, 2015, pp. 2962–2970.

[13] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *PVLDB*, vol. 13, no. 11, pp. 1948–1961, 2020.

[14] K. M. Hazelwood, S. Bird, D. M. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *HPCA*, 2018, pp. 620–629.

[15] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Wrangler: interactive visual specification of data transformation scripts," in *CHI*, 2011, pp. 3363–3372.

[16] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," *J. Open Source Softw.*, vol. 2, no. 19, p. 432, 2017.

[17] S. Schelter, F. Bießmann, D. Lange, T. Rukat, P. Schmidt, S. Seufert, P. Brunelle, and A. Taptunov, "Unit testing data with deequ," in *SIGMOD*, 2019, pp. 1993–1996.

[18] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, "Detecting data errors: Where are we and what needs to be done?" *PVLDB*, vol. 9, no. 12, pp. 993–1004, 2016.

[19] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015, pp. 1247–1261.

[20] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang, "NADEEF: a commodity data cleaning system," in *SIGMOD*, 2013, pp. 541–552.

[21] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, “HoloClean: holistic data repairs with probabilistic inference,” *PVLDB*, vol. 10, no. 11, pp. 1190–1201, 2017.

[22] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, “HoloDetect: few-shot learning for error detection,” in *SIGMOD*, 2019, pp. 829–846.

[23] P. Wang and Y. He, “Uni-Detect: A unified approach to automated error detection in tables,” in *SIGMOD*, 2019, pp. 811–828.

[24] L. Visengeriyeva and Z. Abedjan, “Metadata-driven error detection,” in *SSDBM*, 2018, pp. 1:1–1:12.

[25] M. Mahdavi, Z. Abedjan, R. C. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, “Raha: A configuration-free error detection system,” in *SIGMOD*, 2019, pp. 865–882.

[26] F. Neutatz, M. Mahdavi, and Z. Abedjan, “ED2: A case for active learning in error detection,” in *CIKM*, 2019, pp. 2249–2252.

[27] J. Z. Li, “Principled approaches to robust machine learning and beyond,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, USA, 2018.

[28] Z. Liu, J. Park, N. Palumbo, T. Rekatsinas, and C. Tzamos, “Robust Mean Estimation under Coordinate-level Corruption with Missing Entries,” *arXiv*, 2020.

[29] B. Zhu, J. Jiao, and J. Steinhardt, “When does the tukey median work?” *ISIT*, pp. 1201–1206, 2020.

[30] I. Diakonikolas, G. Kamath, D. Kane, J. Li, J. Steinhardt, and A. Stewart, “Sever: A robust meta-algorithm for stochastic optimization,” in *ICML*, 2019.

[31] V. Raman and J. M. Hellerstein, “Potter’s wheel: An interactive data cleaning system,” in *VLDB*, 2001, pp. 381–390.

[32] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, “Dataxformer: A robust transformation discovery system,” in *ICDE*, 2016, pp. 1134–1145.

[33] (2021) OpenRefine. [Online]. Available: https://openrefine.org/

[34] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, “Continuous data cleaning,” in *ICDE*, 2014, pp. 244–255.

[35] C. Pit-Claudel *et al.*, “Outlier detection in heterogeneous datasets using automatic tuple expansion,” Technical Report MIT-CSAIL-TR-2016-002, CSAIL, MIT, 32 Vassar Street, Cambridge MA 02139, Tech. Rep., 2016.

[36] V. J. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, 2004.

[37] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Bießmann, and A. Grafberger, “Automating large-scale data quality verification,” *PVLDB*, vol. 11, no. 12, pp. 1781–1794, 2018.

[38] E. Breck, N. Polyzotis, S. Roy, S. Whang, and M. Zinkevich, “Data validation for machine learning,” in *MLSys*, 2019.

[39] T. Dasu and J. M. Loh, “Statistical distortion: Consequences of data cleaning,” *PVLDB*, vol. 5, no. 11, pp. 1674–1683, 2012.

[40] F. Bießmann, D. Salinas, S. Schelter, P. Schmidt, and D. Lange, “"Deep" learning for missing value imputation in tables with non-numerical data,” in *CIKM*, 2018, pp. 2017–2025.

[41] F. Bießmann, T. Rukat, P. Schmidt, P. Naidu, S. Schelter, A. Taptunov, D. Lange, and D. Salinas, “DataWig: Missing value imputation for tables,” *J. Mach. Learn. Res.*, vol. 20, pp. 175:1–175:6, 2019.

[42] Z. Liu, Z. Zhou, and T. Rekatsinas, “Picket: Self-supervised data diagnostics for ML pipelines,” *CoRR*, vol. abs/2006.04730, 2020.

[43] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, “ActiveClean: interactive data cleaning for statistical modeling,” *PVLDB*, vol. 9, no. 12, pp. 948–959, 2016.

[44] B. Karlas, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang, “Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions,” *PVLDB*, vol. 14, 2021.

[45] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, “BoostClean: automated error detection and repair for machine learning,” *CoRR*, vol. abs/1711.01299, 2017.

[46] S. Krishnan and E. Wu, “AlphaClean: Automatic generation of data cleaning pipelines,” *CoRR*, vol. abs/1904.11827, 2019.

[47] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden, “Outlier detection in heterogeneous datasets using automatic tuple expansion,” Technical Report MIT-CSAIL-TR-2016-002, CSAIL, MIT, 32 Vassar Street, Cambridge MA 02139, Tech. Rep., 2016.

[48] W. Wu, L. Flokas, E. Wu, and J. Wang, “Complaint-driven training data debugging for query 2.0,” in *SIGMOD*, 2020, pp. 1317–1334.

[49] P. W. Koh and P. Liang, “Understanding black-box predictions via influence functions,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70, 2017, pp. 1885–1894.

[50] C. M. Bishop, “Training with noise is equivalent to tikhonov regularization,” *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[51] D. A. Van Dyk and X.-L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001.

[52] R. R. Wilcox, *Introduction to robust estimation and hypothesis testing*. Academic press, 2011.

[53] J. Steinhardt, *Robust learning: Information theory and algorithms*. Stanford University, 2018.

[54] M. Belkin, D. J. Hsu, and P. Mitra, “Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate,” in *NeurIPS*, 2018, pp. 2306–2317.

[55] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *ICLR*, 2018.

[56] M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, “Certified robustness to adversarial examples with differential privacy,” in *IEEE Symposium on Security and Privacy*, 2019, pp. 656–672.

[57] A. Raghunathan, J. Steinhardt, and P. Liang, “Certified defenses against adversarial examples,” in *ICLR*, 2018.

[58] J. Picado, J. Davis, A. Termehchy, and G. Y. Lee, “Learning over dirty data without cleaning,” in *SIGMOD*, 2020, pp. 1301–1316.

[59] A. Mueller, “Dabl: the data analysis baseline library,” https://dabl.github.io/dev/, 2020.

[60] J. Wang, “Dataprep: The easiest way to prepare data in python,” https://www2.cs.sfu.ca/~jnwang/ppt/DataPrep-Overview-Databricks.pdf, 2021.

[61] “Google facets,” https://pair-code.github.io/facets/, 2020.

[62] E. K. Rezig, M. Ouzzani, A. K. Elmagarmid, W. G. Aref, and M. Stonebraker, “Towards an end-to-end human-centric data cleaning framework,” in *SIGMOD*, 2019, pp. 1:1–1:7.

[63] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu, “Interventional fairness: Causal database repair for algorithmic fairness,” in *SIGMOD*, 2019, pp. 793–810.

[64] J. D. Mackinlay, “Automatic design of graphical presentations,” Ph.D. dissertation, Stanford University, 1987.

[65] J. D. Mackinlay, P. Hanrahan, and C. Stolte, “Show me: Automatic presentation for visual analysis,” *TVCG*, vol. 13, no. 6, pp. 1137–1144, 2007.

[66] X. Wang, X. L. Dong, and A. Meliou, “Data X-Ray: A diagnostic tool for data errors,” in *SIGMOD*, 2015, pp. 1231–1245.

[67] E. Wu and S. Madden, “Scorpion: Explaining away outliers in aggregate queries,” *PVLDB*, vol. 6, no. 8, pp. 553–564, 2013.

[68] Z. Miao, Q. Zeng, C. Li, B. Glavic, O. Kennedy, and S. Roy, “CAPE: explaining outliers by counterbalancing,” *PVLDB*, vol. 12, no. 12, pp. 1806–1809, 2019.

[69] R. Lourenço, J. Freire, and D. E. Shasha, “BugDoc: Algorithms to debug computational processes,” in *SIGMOD*, 2020, pp. 463–478.

[70] M. Brachmann, C. Bautista, S. Castelo, S. Feng, J. Freire, B. Glavic, O. Kennedy, H. Mueller, R. Rampin, W. Spoth, and Y. Yang, “Data debugging and exploration with vizier,” in *SIGMOD*, 2019, pp. 1877–1880.

[71] A. Meliou, W. Gatterbauer, and D. Suciu, “Reverse data management,” *PVLDB*, vol. 4, no. 12, pp. 1490–1493, 2011.

[72] Y. Wu, E. Dobriban, and S. B. Davidson, “DeltaGrad: Rapid retraining of machine learning models,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 10 355–10 366.

[73] Y. Wu, V. Tannen, and S. B. Davidson, “Priu: A provenance-based approach for incrementally updating regression models,” in *SIGMOD*, 2020, pp. 447–462.

[74] D. Xin, S. Macke, L. Ma, J. Liu, S. Song, and A. G. Parameswaran, “Helix: Holistic optimization for accelerating iterative machine learning,” *PVLDB*, vol. 12, no. 4, pp. 446–460, 2018.

[75] M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthör, K. Innerebner, F. Klezin, S. N. Lindstaedt, A. Phani, B. Rath, B. Reinwald, S. Siddiqui, and S. B. Wrede, “SystemDS: A declarative machine learning system for the end-to-end data science lifecycle,” in *CIDR*, 2020.

[76] B. Lockhart, J. Peng, W. Wu, J. Wang, and E. Wu, “Explaining inference queries with bayesian optimization,” 2021.

[77] K. Ramachandra, K. Park, K. V. Emani, A. Halverson, C. A. Galindo-Legaria, and C. Cunningham, “Froid: Optimization of imperative programs in a relational database,” *PVLDB*, vol. 11, no. 4, pp. 432–444, 2017.

[78] L. Ramjit, M. Interlandi, E. Wu, and R. Netravali, "Acorn: Aggressive result caching in distributed data processing frameworks," in *SoCC*, 2019, pp. 206–219.

[79] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[80] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. M. Trinidad, and J. Kittler, "A review of instance selection methods," *Artif. Intell. Rev.*, vol. 34, no. 2, pp. 133–143, 2010.

[81] S. Redyuk, S. Schelter, T. Rukat, V. Markl, and F. Bießmann, "Learning to validate the predictions of black box machine learning models on unseen data," in *SIGMOD*, 2019, pp. 4:1–4:4.

[82] S. Redyuk, Z. Kaoudi, V. Markl, and S. Schelter, "Automating data quality validation for dynamic data ingestion," in *EDBT*, 2021.

[83] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *NeurIPS*, 2015, pp. 2503–2511.

[84] Z. C. Lipton, Y. Wang, and A. J. Smola, "Detecting and correcting for label shift with black box predictors," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 3128–3136.

[85] H. Abdelkader, "Towards robust production machine learning systems: Managing dataset shift," in *ASE*, 2020, pp. 1164–1166.

[86] P. W. Koh, K. Ang, H. H. K. Teo, and P. Liang, "On the accuracy of influence functions for measuring group effects," in *NeurIPS*, 2019, pp. 5255–5265.

[87] S. Grafberger, J. Stoyanovich, and S. Schelter, "Lightweight inspection of data preprocessing in native machine learning pipelines," in *CIDR*, 2021.

[88] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer, "Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification," *PVLDB*, vol. 13, no. 11, pp. 2508–2521, 2020.

[89] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting training data from large language models," *CoRR*, vol. abs/2012.07805, 2020.

[90] S. Schelter, ""Amnesia" - machine learning models that can forget user data very fast," in *CIDR*, 2020.

[91] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *NeurIPS*, 2018, pp. 7167–7177.

[92] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking clever hans predictors and assessing what machines really learn," *Nature communications*, vol. 10, no. 1, pp. 1–8, 2019.

Validating Data and Models in Continuous ML pipelines

Mike Dreves, Gene Huang, Zhuo Peng, Neoklis Polyzotis, Evan Rosen, Paul Suganthan G. C.
Google Inc.

Abstract

Production ML is more than writing the code for the trainer. It requires processes and tooling that enable a larger team to share, track, analyze, and monitor not only the code for ML but also the artifacts (Datasets, Models, ...) that are manipulated and generated in these production ML pipelines.

In this paper we describe the tools we developed at Google for the analysis and validation of two of the most important types of artifacts: Datasets and Models. These tools are currently deployed in production at Google and other large organizations. Our approach is heavily inspired by well-known principles of data-management systems. Ultimately, we want to enable users to trust their data and models, and understand how data properties affect the quality of the generated ML models.

1 Introduction

ML has succeeded in tackling a wide range of challenging problems in practice, from medical applications (e.g., the detection of retinopathy [1]), to self-driving cars, or agriculture [3], to name a few cases. Moreover, there is a fast pace of innovation in the scientific field of ML.

When one begins to use ML, they naturally think about writing the code that can train a model based on input data. This is an important step of using ML, but this code is only a small piece of what it takes to run ML reliably. In a production setting, the user of ML has to worry about: whether the input data has errors; how to trigger a retraining of the model when new data arrives; whether the new version of a model is good enough to replace the model currently used by the downstream stack; whether the serving data (used for prediction requests to the model) is sufficiently different such that retraining is required; and many more issues.

At Google, we refer to this set of concerns as "ML engineering" in order to separate them from "ML coding", which refers to the relatively smaller task of writing the trainer. In using this term we draw an analogy to the distinction between software engineering and coding. Usually, coding refers to a one-off, monolithic piece of code that is not meant for sharing, whereas software engineering establishes processes and practices around versioning, testing, and so on, to enable a larger team to work on a shared code base that solves a bigger problem. The same applies to ML engineering: we need processes and tooling around versioning, testing, monitoring, and so on, that apply not only to code but also to ML artifacts such as datasets and models.

In this paper, we describe the tooling and processes we developed for the analysis and validation of datasets and models. These are two of the most important artifacts in production ML pipelines, with clear connections to the overall effectiveness of ML. We adopt a data-oriented approach to the management of these artifacts. This is obvious for dataset artifacts. For model artifacts, we note that the quality of a model is inherently tied to the data

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

used for training, evaluation, and serving. Moreover, with the advent and maturation of Auto-ML solutions which can optimize the parameters of the model's architecture, the input data becomes the next obvious knob that can be tweaked to improve model quality. Thus, an approach based on data-management principles can be usefully extended to models.

Our work is in the context of TensorFlow Extended (TFX), a production-ML platform that we developed for Google's product teams and the world. The design and functionality of our tools is heavily influenced by our experience with ML in Google's production setting, but we believe that the basic principles and lessons learned apply to other contexts, and are of interest to both researchers and practitioners.

2 Data Validation

ML crucially depends on the quality of the input data in order to produce a good model. In many cases, even simple errors in the data can affect significantly the resulting model. For example, consider a model that depends on an input data feature "Country" where the USA is represented with the string "US". However, if in the next batch of training data the value becomes "USA", then without any validation or pre-processing, the model will simply think that this is a new country.

Furthermore, it is often the case that the predictions from the generated models are logged and used to generate more data for training. Such feedback loops can amplify even "small" data errors and lead to gradual regression of model performance over a period of time. Hence, it is critical to catch data errors early, before they propagate through the complex loops and taint more of the pipeline's state.

Finally, error-free data is also critical for model understanding, since any attempt to debug and understand the output of the model or any quality metrics (e.g., AUC, precision, ...) is futile if the evaluation data has errors. All these observations indicate that we need to treat data as a first-class citizen in ML pipelines, on par with algorithms and infrastructure, with corresponding tooling to analyze, validate and monitor the data throughout the various stages of the pipeline.

Data validation is neither a new problem nor unique to ML, and so we can leverage techniques and principles from the field of data management. However, we argue that the problem acquires unique characteristics in the context of ML and hence we need to rethink existing solutions. First, we need a way to express ML-related constraints and expectations on the quality of the data. Second, the data validation system must generate reliable alerts with high precision, and provide enough context for the human to quickly identify the root cause of the problem. This is due to the fact that in most cases data errors cannot be fixed automatically – they require some human intervention, either in the ML pipeline (e.g., rolling back the trainer to a checkpoint unaffected by the suspect data) or in the data-generation code (e.g., fixing the bugs that cause the errors). Third, the system needs to scale to production pipelines which typically process billions to trillions of examples. Finally, the system needs to account for the fact that data is stored and managed externally from the ML pipeline, and often in a variety of storage systems, and hence a-priori knowledge about the data and its semantics is limited.

To address the above challenges in the context of Google's production ML pipelines, we developed TensorFlow Data Validation (TFDV) [13, 12, 6], a scalable data analysis and validation system for ML. Our system is deployed in production as an integral part of TFX [10], an end-to-end ML platform, and is used by hundreds of product teams at Google to monitor and validate trillions of training and serving examples per day, amounting to several petabytes of data per day. We recently open sourced TFDV and the system has received significant attention from the open-source community as well: more than 50M downloads since the first release in October 2018, plus it has influenced the development of other open-source data validation systems such as Apache Spark Data Validation [1]. Furthermore, TFDV has been adopted by other large organizations using ML, e.g., see Spotify's keynote at TensorFlow World 2019 about using TFDV [2].

[1]https://databricks.com/session/apache-spark-data-validation

[2]https://www.youtube.com/watch?v=zxd3Q2gdArY&t=748s

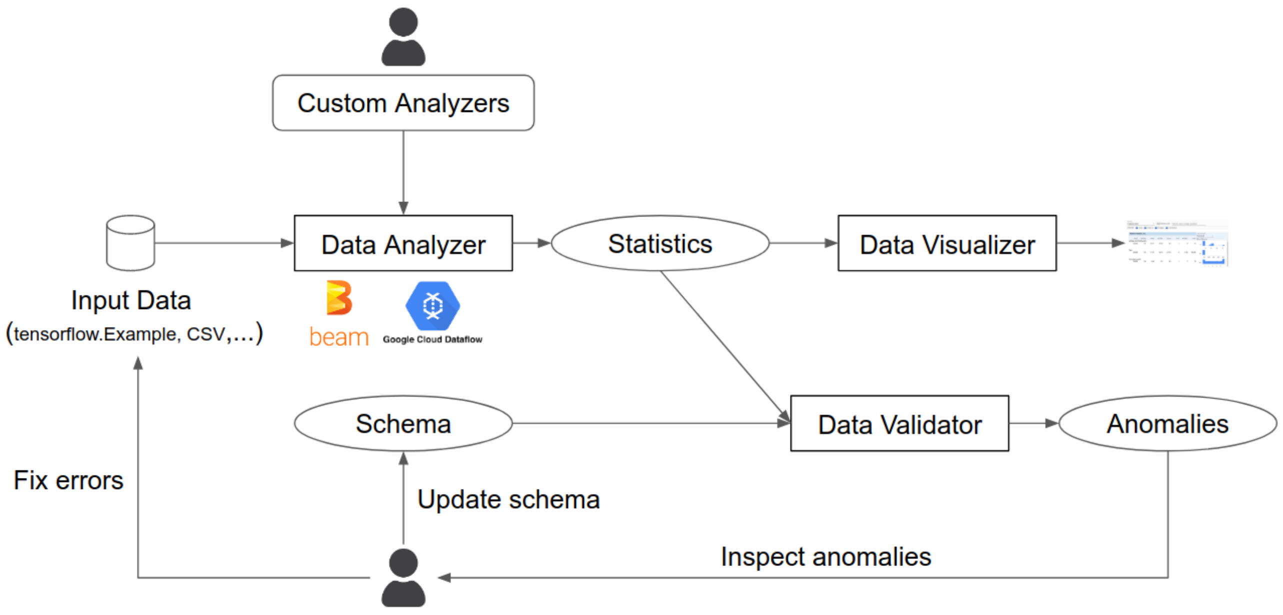


Figure 1: TensorFlow Data Validation Architecture.

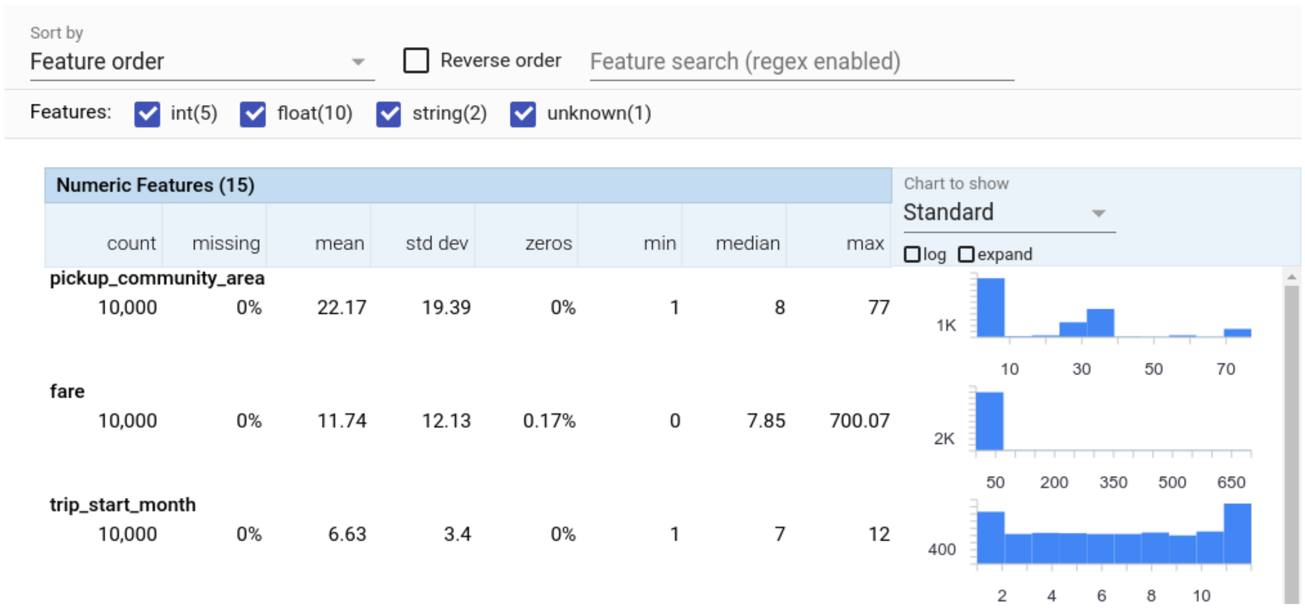
Few recent works ([19], [18]) have considered the importance of data validation for ML applications. For example, Amazon (Schelter et. al. [19]) proposed a system for automating data quality verification task that provides a declarative API to specify common quality constraints and custom validation code. While Amazon's system allows users to express arbitrary constraints, we opted to have a restrictive schema definition language that captures the data constraints for most of our users in order to focus on reliable high precision alerts. Further, our work emphasizes the user-mediated co-evolution of the schema with data and the model. As far as we can tell, TFDV is the first open-source data analysis and validation system for ML.

2.1 System Overview

Figure 1 shows the TFDV architecture which consists of a Data Analyzer component which computes statistics in a scalable fashion over large amounts of data, a Data Validator which finds anomalies in the data, and a Data Visualizer which provides visualizations of the statistics, schema, and the anomalies.

Data Analyzer. Data Analyzer takes a collection of statistics generators and computes data statistics needed for validation. TFDV uses Apache Beam [8] to define and process its data pipelines. The statistics generators are implemented as Beam transforms. Users can provide custom statistics generators which are executed together with the default generators. The generator API takes Apache Arrow Tables as input, as it is powerful enough to encode popular logical training data formats: flat (tensorflow.Example, CSV), sequence (tensorflow.SequenceExample) or structured data (e.g. Protocol Buffers or Apache Avro). TFDV provides decoders for popular data formats like tensorflow.Example, and CSV. Users can write custom decoders (that convert their input to Arrow Tables) to handle arbitrary data formats. Realizing the data pipelines using Beam allows TFDV to transparently run the pipeline in different environments such as a single machine, Flink/Spark cluster, and Google Cloud Dataflow.

The statistics computed by TFDV include individual feature statistics depending on the type of the feature (e.g., statistics such as min, max, mean, median, histogram etc. for numeric features, statistics such as number of unique values, top-k values, average length etc. for categorical features.) and cross-feature statistics (e.g., correlation between features and mutual information of a feature with the label etc.). We represent the statistics as a protocol buffer message (See [9] for the complete list of statistics computed by TFDV.).


Sort by: Feature order

☐ Reverse order

Feature search (regex enabled)

Features: ✓ int(5) ✓ float(10) ✓ string(2) ✓ unknown(1)

Numeric Features (15)

	count	missing	mean	std dev	zeros	min	median	max
pickup_community_area	10,000	0%	22.17	19.39	0%	1	8	77
fare	10,000	0%	11.74	12.13	0.17%	0	7.85	700.07
trip_start_month	10,000	0%	6.63	3.4	0%	1	7	12

Chart to show: Standard

☐log ☐expand

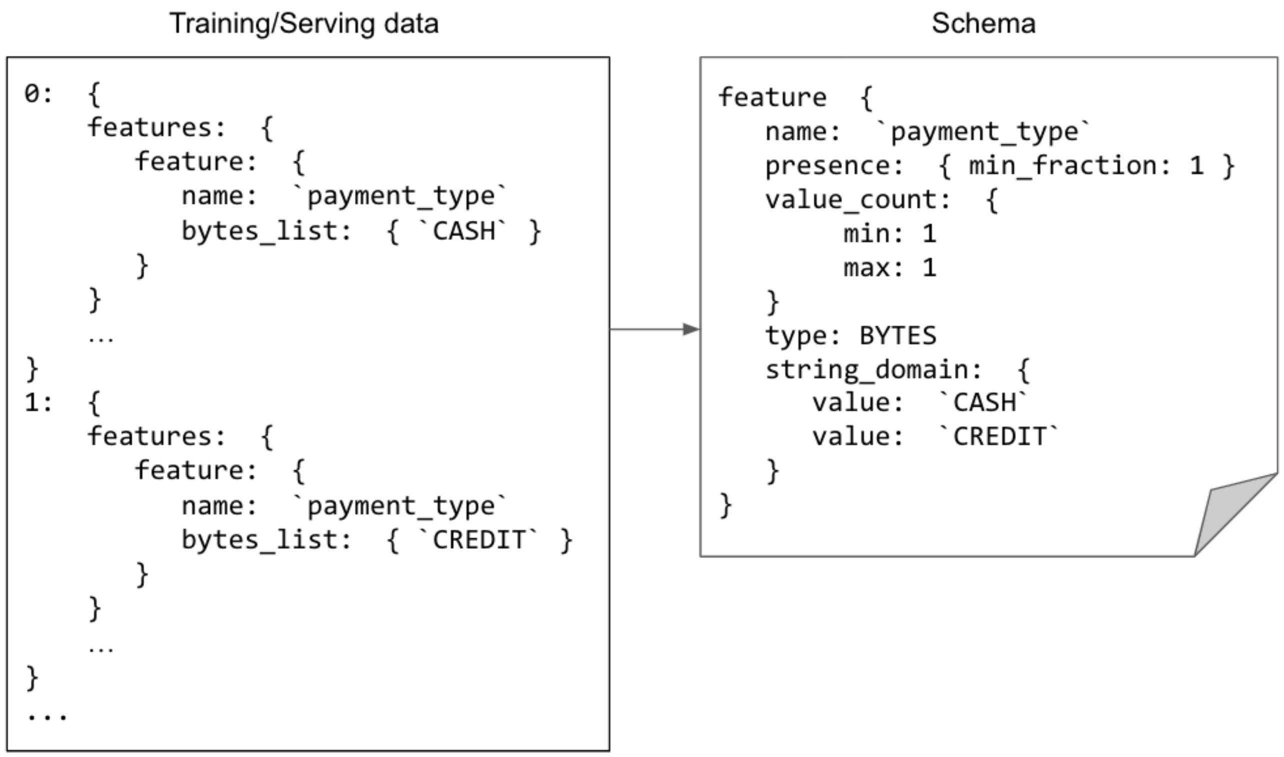
Figure 2: Exploring data using Facets visualization.

Data Validator. The Data Validator checks the properties of the data as specified through a schema. Typically we expect the data characteristics to remain stable across different splits of data (e.g., training data and evaluation data) or batches of data that are close in time. Hence, we consider any deviation within a batch (or split) from the expected data characteristics as an *anomaly*.

TFDV adapts the "battle-tested" principles from data management systems to the context of ML. Specifically, in order to codify these expected data characteristics, TFDV generalizes the traditional notion of a *schema* from database systems. The schema follows a logical data model where each training or serving example is a collection of features, with each feature having several constraints attached to it. This flat model has an obvious mapping to the flat data formats such as tensorflow.Example or CSV. The constraints associated with each feature cover some basic properties (e.g., type, domain, valency) but also constraints that are relevant to ML (See [12] for a more detailed discussion of our schema formalism.). Using a schema also allows us to verify any assumptions of training/serving code (e.g., the schema can be used to generate fuzzy examples and verify if the training/serving code crashes on those examples) and thereby catch potential model crashes early on. We represent the schema as a protocol buffer message.

TFDV supports two types of validation: (1) validating a single batch of data against the schema, and (2) validating two batches of data (e.g., are there any significant changes between training and serving data, or between successive batches of the training data?). Any disagreement found during validation is flagged as an anomaly for human inspection and further investigation. See [6] for the complete list of 52 anomalies identified by TFDV and the conditions on which each anomaly is raised.

Data Visualizer. TFDV provides visualizations for the statistics, schema and the anomalies. It provides a simple table-based view for the schema and the anomalies. It uses the Facets library [7] to visualize the statistics (see Figure 2). Specifically, TFDV supports (1) visualizing the statistics of a batch of data, and (2) comparing statistics between batches of data.



```
0:  {
    features:  {
       feature:  {
          name:  `payment_type`
          bytes_list:  { `CASH` }
       }
    }
    ...
}
1:  {
    features:  {
       feature:  {
          name:  `payment_type`
          bytes_list:  { `CREDIT` }
       }
    }
    ...
}
...
```

```
feature  {
   name:  `payment_type`
   presence:  { min_fraction: 1 }
   value_count:  {
        min: 1
        max: 1
   }
   type: BYTES
   string_domain:  {
      value:  `CASH`
      value:  `CREDIT`
   }
}
```

Figure 3: An example schema and corresponding data in the tf.train.Example format.

2.2 Inferring an initial schema

As described in Section 2.1, TFDV uses a schema to capture the stable data characteristics. Our assumption is that the users are responsible to curate the schema. However, many ML pipelines use thousands of features, and so constructing a schema manually for such pipelines can be quite tedious. Furthermore, the domain knowledge of the features may be distributed across a number of engineers within the product team or even outside of the team. In such cases, the upfront effort in schema construction can discourage engineers from setting up data validation until they run into a serious data error.

To overcome this adoption hurdle, TFDV provides a way to auto-generate the initial schema. This auto-generated schema attempts to capture salient properties of the data without overfitting to a particular batch of data. Avoiding overfitting is important: an overfitted schema is more likely to cause spurious alerts when validating a new batch of data, which in turn increases the cognitive overhead for the on-call engineers, reduces their trust in the system, and may even lead them to switch off data validation altogether. We currently rely on a set of reasonable heuristics to perform this initial schema inference. A more formal solution, perhaps with guarantees or controls on the amount of overfitting, is an interesting direction for future work. We assume the user will inspect the inferred schema and modify any properties if needed. Figure 3 shows a sample schema. The schema is represented as a protocol buffer.

2.3 Performing skew detection

Training/Serving skew refers to a difference in the feature values or distributions between the data used to train a model and the data observed by the serving system. TFDV allows users to check if there are significant changes between training and serving data. TFDV supports skew detection based on L_∞ distance [12] and Jensen–Shannon divergence [2].

3 Model Validation

After training a model based on (validated) input data, the next step in a production pipeline is analyzing the trained model and deciding whether it can be pushed to the inference/serving stack. This step makes the model available to other applications. However, pushing a model that returns sub-optimal or even erroneous predictions can lead to undesired downstream effects. As an example, if a bad model generates uninteresting or irrelevant recommendations in a movie-streaming service the user experience is likely to suffer. What is needed is the equivalent of an integration test, where we can evaluate the model's accuracy on unseen data. For this purpose, we have developed TensorFlow Model Analysis (TFMA) [4], a scalable model evaluation system. Similar to TFDV TFMA is deployed in production as an integral part of TFX platform and is used by hundreds of product teams at Google to evaluate ML models. TFMA was open sourced in March 2018 and has received significant attention from the open-source community (more than 50M downloads).

While data validation prevents errors through inappropriate input data, it is still possible to introduce errors into the pipeline as a result of improper training. Most ML frameworks provide tools for evaluating metrics of interest (e.g., loss or AUC) during training. TFMA takes this a step further by allowing users to re-evaluate their models post-training over large amounts of data in a distributed manner (using Apache Beam). This evaluation can be done using the same metrics that were defined during training or with additional metrics added after the model has been generated. While TFMA performs full passes over generally much larger data than is seen at training time, the datasets are still just samples drawn from a larger population. To help gauge the reliability of these computations, TFMA provides confidence intervals for the metrics it computes. For increased reliability and safety, TFMA provides a means to set expectations on model performance metrics using either absolute thresholds or thresholds relative to a baseline. Practitioners can then gate pushing their models to production based on passing the said validation thresholds. In a sense, this model-based validation complements the data-based validation implemented by TFDV.

TFMA also supports computing and validating model metrics on data slices. Model metrics computed on the whole evaluation dataset can mask interesting or significant deviations of the same metrics computed on data slices that correspond to meaningful sub-populations. For instance, a machine-translation model may perform adequately well on average but significantly worse on a specific language. TFMA allows users to declare slices of interest and then to get a more detailed view of model metrics computed on the corresponding subsets of the evaluation data. Model validation can subsequently verify these metrics in order to guard against model regressions which only affect a small but important slice of the evaluation data.

3.1 System Overview

Figure 4 shows the TFMA architecture which consists of four components: 1) Reading the inputs, 2) Extraction, 3) Evaluation, and 4) Writing results. These components make use of two primary types: tfma.Extract and tfma.evaluators.Evaluation. The type tfma.Extract represents the data that is extracted during pipeline processing whereas the type tfma.evaluators.Evaluation represents the output from evaluating the extracts at various points during the pipeline. In order to provide a flexible API, these types are just Python dictionaries where the keys are defined (reserved for use) by different implementations.

Extraction. The extraction process is a list of Beam transforms that are run in series. The extractors take tfma.Extract as input and return tfma.Extract as output. For example, PredictExtractor is one of the default extractors which uses the input extract produced by the read inputs transform and runs it through a model to produce extracts that contains the predictions. TFMA allows users to provide custom extractors that can be inserted at any point in the extraction process.

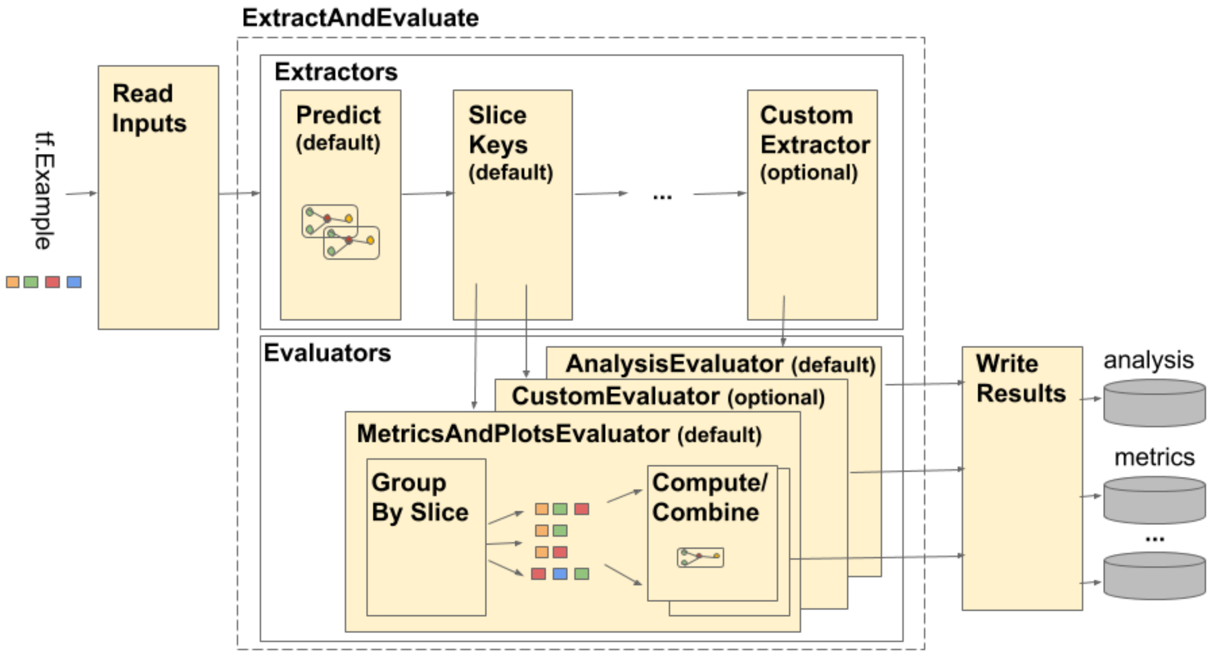
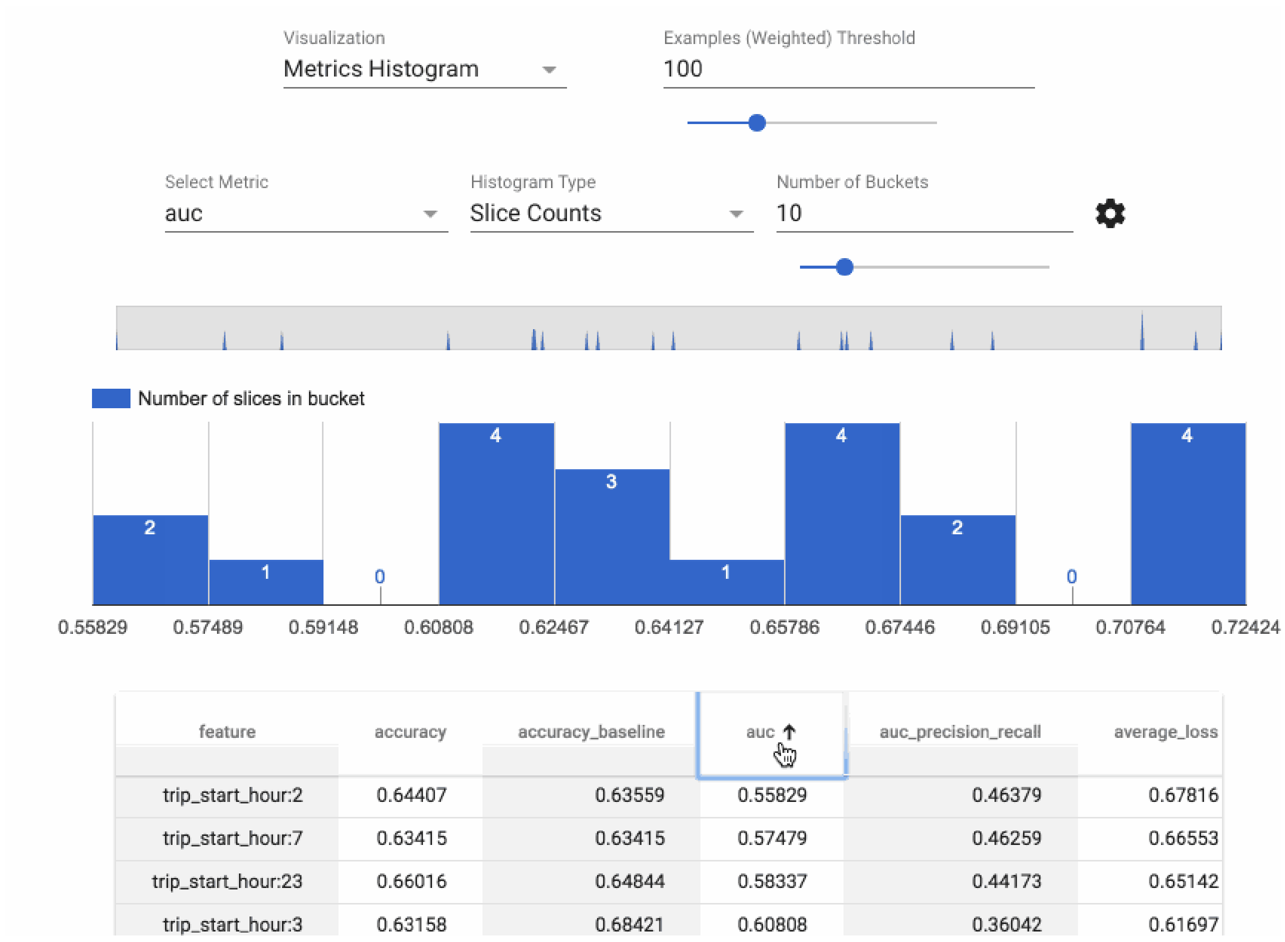


Figure 4: TensorFlow Model Analysis Architecture.

Evaluation. Evaluation is the process of taking an extract and evaluating it. An evaluator is a Beam transform that takes tfma.Extract as inputs and outputs tfma.evaluators.Evaluation. TFMA supports a wide variety of model metrics and plots as documented in [5], which are computed as part of the evaluation process. For example, the default MetricsAndPlotsEvaluator uses the features, labels, and predictions in the extracts as input, group the extracts by slices, and then performs metrics and plots computations. Further, users can also provide custom evaluators. TFMA provides visualizations for the metrics and plots as shown in Figure 5.

3.2 Automatically Identifying Problematic Slices

As mentioned in Section 3, TFMA can compute model metrics over slices of data that are of interest to the user. For example, Figure 5 shows the model metrics sliced by “trip_start_hour" feature. However in many cases, users do not know a priori which slices are important. Alternatively, a user might miss important sub-slices within the manually defined slices (e.g., the machine-translation model may under-perform for a specific language only when the translation request comes from a specific class of devices). To aid with these cases we investigated techniques to automatically identify slices of interest [11]. We view this automatic slicing as a complement to the manual slicing already supported by TFMA. For instance, it is possible to start with manually defined slices and then automatically discover subslices of interest. Our inspiration in this domain has been previous works in discovering interesting roll-ups/drill-downs in data cubes [14, 15, 17]. Moreover, recent work [16] in the ML community has shown how knowledge about under-performing slices can be leveraged in order to improve overall model quality.



Visualization: Metrics Histogram
Examples (Weighted) Threshold: 100
Select Metric: auc
Histogram Type: Slice Counts
Number of Buckets: 10

feature	accuracy	accuracy_baseline	auc ↑	auc_precision_recall	average_loss
trip_start_hour:2	0.64407	0.63559	0.55829	0.46379	0.67816
trip_start_hour:7	0.63415	0.63415	0.57479	0.46259	0.66553
trip_start_hour:23	0.66016	0.64844	0.58337	0.44173	0.65142
trip_start_hour:3	0.63158	0.68421	0.60808	0.36042	0.61697

Figure 5: Visualization of sliced model metrics.

4 Conclusion

In this paper, we described the tools and processes developed at Google for the analysis and validation of two of the most important artifacts in an ML pipeline: *datasets* and *models*. Specifically, we described TFDV, a data validation system, and TFMA, a model evaluation system. The developed tools have been used extensively within Google and has received significant attention from the open-source community.

References

[1] Diagnosing Diabetic Retinopathy with Machine Learning. `https://about.google/intl/en_us/stories/seeingpotential`

[2] Jensen–Shannon divergence. `https://en.wikipedia.org/wiki/Jensen-Shannon_divergence`

[3] How a Japanese cucumber farmer is using deep learning and TensorFlow. `https://cloud.google.com/blog/products/gcp/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow`

[4] TensorFlow Model Analysis. `https://www.tensorflow.org/tfx/model_analysis/get_started`

[5] TensorFlow Model Analysis - Metrics and Plots. `https://www.tensorflow.org/tfx/model_analysis/metrics`

[6] TensorFlow Data Validation. `https://www.tensorflow.org/tfx/data_validation/get_started`

[7] Facets. `https://pair-code.github.io/facets/`

[8] Apache Beam. `https://beam.apache.org/`

[9] TensorFlow Metadata. `https://github.com/tensorflow/metadata`

[10] Denis Baylor, et al. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *ACM SIGKDD*, 2017.

[11] Yeounoh Chung, et al. Slice Finder: Automated Data Slicing for Model Validation. In *IEEE ICDE*, pages 1550–1553, 2019.

[12] Eric Breck, et al. Data Validation for Machine Learning. In *SysML*, 2019.

[13] Emily Caveness, et al. TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines. In *ACM SIGMOD Demo*, 2020.

[14] Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, volume 2000, pages 307–316, 2000.

[15] Gayatri Sathe, and Sunita Sarawagi. Intelligent rollups in multidimensional OLAP data. In *VLDB*, volume 1, pages 531–540, 2001.

[16] Vincent Chen, et al. Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices. In *Advances in Neural Information Processing Systems*, pages 9397–9407, 2019.

[17] Manasi Vartak, et al.. SeeDB: Automatically Generating Query Visualizations. In *PVLDB*, volume 7, number 13, pages 1581–1584, 2014.

[18] Rabanser, Stephan and Günnemann, Stephan and Lipton, Zachary. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *Advances in Neural Information Processing Systems*, 2019.

[19] Sebastian Schelter, et al. Automating Large-Scale Data Quality Verification. In *VLDB*, volume 11, number 12, pages 1781–1794, 2018.

Automated Data Validation in Machine Learning Systems

Felix Biessmann, Jacek Golebiowski, Tammo Rukat, Dustin Lange, Philipp Schmidt
{biessman,jacekgo,tammruka,langed,phschmid}@amazon.com
Amazon

Abstract

Machine Learning (ML) algorithms are a standard component of modern software systems. The validation of data ingested and produced by ML components has become a central challenge in the deployment and maintenance of ML systems. Subtle changes in the input data can result in unpredictable behavior of an ML algorithm that can lead to unreliable or unfair ML predictions. Responsible usage of ML components thus requires well calibrated and scalable data validation systems. Here, we highlight some challenges associated with data validation in ML systems. We review some of the solutions developed to validate data at the various stages of a data pipeline in modern ML systems, discuss their strengths and weaknesses and assess to what extent these solutions are being used in practice. The research reviewed indicates that the increasing need for data validation in ML systems has driven enormous progress in an emerging community of ML and Data Base Management Systems (DBMS) researchers. While this research has led to a number of technical solutions we find that many ML systems deployed in industrial applications are not leveraging the full potential of data validation in practice. The reasons for this are not only technical challenges, but there are also cultural, ethical and legal aspects that need to be taken into account when building data validation solutions for ML systems. We identify the lack of automation in data validation as one of the key factors slowing down adoption of validation solutions and translation of research into useful and robust ML applications. We conclude with an outlook on research directions at the intersection of ML and DBMS research to improve the development, deployment and maintenance of ML systems.

1 Introduction

Machine Learning (ML) technology has become a standard component in modern software systems. Many decisions are increasingly being automated with ML and the predictions of ML models are being exposed in data products or consumed by other downstream software components. This trend gives rise to new research challenges at the intersection between Data Base Management Systems (DBMS) community and the ML community. Many of these challenges are related to data validation. In contrast to standard software systems, for which a large arsenal of testing concepts and utilities exists, testing of ML systems is difficult. Depending on the ingested data, ML systems can behave very different, and often subtle changes in the input data, that are hard to detect by humans, can lead to very different ML predictions [4].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This data dependency in the transformations induced by ML models is their very strength: It allows these systems to adapt to any problem setting by learning from rules defined implicitly by a data set. But this flexibility comes at a cost: The more complex and powerful ML systems become, the more difficult it becomes to validate their predictions.

For decades ML scientists have been considering data validation rather an engineering challenge than a research problem. Recently however with more ML systems being deployed in customer facing applications newspaper headlines remind us that without proper validation of ML components we will see more racist chatbots[1] or fatal car crashes[2]. Scientists in the field are beginning to take this problem very seriously. There is an emergent field of research around data management tailored to ML workflows [36]. Testing, monitoring and validation of ML systems and the data ingested and produced by ML models has become a major focus of research with dedicated workshops at major conferences and a rapidly growing research community, as summarized in section 3. Also the recent trend in ML to render models and their predictions more transparent [52] can be regarded as an attempt to validate ML systems and training or prediction data [67]. However to the best of our knowledge there is no commonly agreed upon data validation solution for ML systems that has reached broad adoption. Here, we argue that one of the most important factors is that *automating* validation of ML systems is difficult. Especially when systems learn autonomously and continuously, it can be challenging to ensure that the performance of an ML system is not shifting due to accidental or adversarial changes in the data. Given the above examples for how ML systems can fail in practice, it is obvious that ML systems require scalable, robust and automated data validation solutions. This constraint does not apply to academic research, and thus the lack of automation in ML system validation can be considered as a major blocker slowing down the transfer of the often rapidly evolving advances in ML research into robust and trustworthy customer facing products.

But why is this so difficult to automate? Researchers in the DBMS community have been investigating data profiling for decades [3] and many of these solutions are just right for some aspects of data validation also in the context of ML systems. However some of the challenges in ML system monitoring go beyond that. Many data validation aspects in ML systems *depend on the state of the trained ML model*, such as the performance of a ML model under data set shift, or the differential privacy of a model [16, 2, 5]. Other aspects such as fairness of a ML model require domain expertise that ML engineers often do not have. As illustrated in Figure 1 the competencies required to validate data in the various stages of an ML system require competencies that are usually distributed across several experts in a team or even separate teams.

In the following chapters we will first review the challenges associated with data validation in ML systems and highlighting some of the practical implications. Afterwards we review some of the data validation solutions, with a special focus on the practical applicability of these approaches. Finally we will conclude with an outlook of technical and non-technical challenges associated with ML system validation in order to ensure more responsible usage of ML systems in production software systems.

2 Data Validation Dimensions

As ML systems learn from data, validation of the data ingested during training and prediction is a fundamental prerequisite for well functioning and robust ML systems [58, 10]. Relevant data validation dimensions for ML systems can be broadly categorized into those notions of data quality commonly considered in classical data base management systems (DBMS) and ML model dependent dimensions. In the following, we highlight some of the dimensions and provide examples for each category.

[1]`https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/`

[2]`https://www.nytimes.com/2016/09/15/business/fatal-tesla-crash-in-china-involved-autopilot-government-tv-says.html`

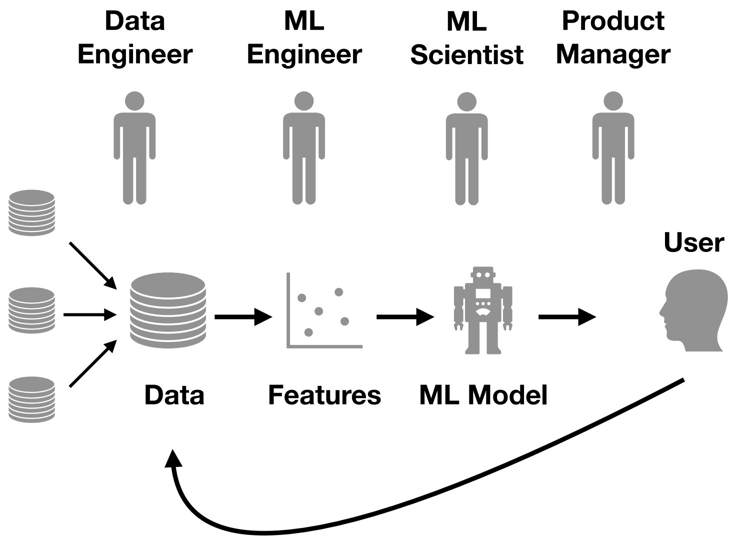


Figure 1: Responsibilities for a machine learning (ML) production software system are often distributed across different roles and teams [1]. Data from various sources is integrated, features are extracted and ML models transform these into predictions, which are then served to users whose behavioural signals are fed back to the system. Monitoring and validating these data streams at scale is only feasible with automated tests. But validating the data in each of these stages requires different competencies and often domain expertise. This imposes novel technical and conceptual challenges on engineering teams when automating ML production systems.

2.1 Classical DBMS dimensions

Independent of ML applications there are many data validation dimensions that have been investigated in the DBMS community, more specifically in the data profiling literature[3]:

Data Correctness: This dimension refers to schema violations, such as string values in numeric columns, or generally syntactically wrong data. Defects in this dimension can often break ML processing pipelines. Some of these violations can be detected easily for instance by type checks, and corrupted data instances can be filtered out.

Data Consistency refers to defects that are not captured by syntactic correctness, including duplicate entries or invalid values. Detecting these cases can be difficult and computationally challenging, but there exist efficient approaches to de-duplication and detection of semantic inconsistencies. Violations of semantic inconsistencies can for instance be detected by validation of functional or approximate functional dependencies [46, 35].

Completeness of a data source reflects the ratio of missing values in a data set. In principle this dimension is simple to probe, however only under the assumption that the missing value symbol is known. Unfortunately this assumption is often violated. As missing values often cause problems in data pipelines feeding into ML systems, a common strategy of avoid such problems is to replace missing values with syntactically valid values. These transformations often make the missing values pass validations for correctness and consistency.

Statistical Properties: This dimension includes basic descriptive statistics of an attribute, such as mean of a numeric or mode of a categorical variable, and more complex parametric or non-parametric statistical aggregates of a data set. A popular validation strategy in this data dimension is anomaly detection.

2.2 ML model dependent dimensions

Validating the above data quality dimensions can be challenging, but these tests of the input data are independent of the downstream ML model. This implies that testing only for these dimensions can lead to both false negatives, when a data corruption is not detected that has a negative impact on the downstream ML model, as well as false positives, when a data corruption raises an alarm but does not have any effect on the downstream ML model. Examples for false negatives of ML model independent validations include adversarial attacks, such as for instance avoiding filters for adult language by *leet speak*[3] strings obfuscated by misspellings and replacing letters with numeric characters. Examples for false positive alarms are ML models employing strong L_1 or sparsity-enforcing regularization constraints. In these models, a large number of input features is often not affecting the ML model output, hence distributional shifts in these input features would not be reflected in shifts of the ML model outputs and validation of the input data independent of the model outputs will raise false alarms.

Data validation in the context of ML production systems thus has to take into account the current state of the downstream ML model to ensure robust and reliable performance. The state of the ML model and and the impact of data corruptions on ML performance can be monitored by a number of different validation dimensions of the ML model *output* data. These have to be validated along with the usual criteria common to all production software systems, such as requirements on the latency and availability of the system. SageMaker Model Monitor[4] is a solution that enables monitoring of models deployed in production. For example, alarms are triggered when data distributions shifts are detected.

Predictive performance metrics are usually the most important metrics optimized in production ML systems. These metrics include accuracy, precision or F1-score of a classifier, the mean-squared error, mean absolute error or r^2-scores for regression models or various ranking losses, evaluated via cross-validation on a test set that was not used for training the respective ML model. When the data processed by trained and deployed ML model is drawn from the same distribution as the training and test data, then these cross-validated metrics reflect the predictive performance reliably. But in production ML systems, this assumption is often violated. Examples include shifts in the input data (covariate shifts) or shifts in the target data distribution (label shift).

Robustness The shifts induced by corruptions in the data that are a) not caught by upstream classical data validation techniques and that have b) a negative impact on the predictive performance of a production ML system can be difficult to detect, as there is no ground truth label information available for the data. In the absence of ground truth data needed to compute predictive performance metrics, one can resort to classical data validation techniques applied to the outputs of ML models. But this approach can fail to detect certain covariate shifts induced by outliers, adversarial attacks or other corruptions.

Privacy metrics have become increasingly important since ML models are used to process personal data such as health care data, financial transactions, movement patterns or really almost every aspect of our lives. Popular concepts include *differential privacy* [17] and *k-anonymity* [61]. Note that k-anonymization [61] is not model dependent, it is a property of a data base where user data is considered private if information about each user cannot be distinguished from at least k-1 other users whose information is in the dataset. In the field of ML, differential privacy is more useful than k-anonymity. Two main reasons for that are a) k-anonymity is defined for

[3]`https://en.wikipedia.org/wiki/Leet`
[4]`https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html`

a fixed data set, if new rows are added, one might need to convert the data again in order to achieve k-anonymity, and b) if a k-anonymous data set is combined with other information, it is often possible to de-anonymize the data. A large body of literature in the ML community builds upon the definition of ϵ-differential privacy [18] which deals with the problem of learning little to nothing about individual data-points while learning a lot about the population by ensuring the same conclusions can be drawn whether any single data-point is included in the dataset or not. Most implementations of differentially private ML models add noise to the data, gradients or the model coefficients [2, 59, 47, 42]. This can be an effective countermeasure to model inversion attacks [20] or membership inference attacks [59]. The amount of noise added determines the amount of privacy preserved – but it also bounds the predictive performance or *utility* of the ML model [30].

Robustness to adversarial data Recent developments have shown that ML models, placed at the centre of security-critical systems such as self-driving cars, can be targeted by dedicated adversarial examples - inputs that are very difficult to distinguish from regular data but are misclassified by the model [22]. Those inputs, created by applying a small but very significant perturbation to regular examples can be used to manipulate the model to predict outputs desired by an adversary [62, 9, 44]. The robustness to adversarial attacks of the model can be measured by (1) generating a set of adversarial examples based on the test set [62, 37, 28, 43, 22] and (2) computing what is the change in models accuracy given a fixed average distortion, or what is the minimum average distortion needed to reach 0% accuracy on the test set.

Fairness Another dimension that is highly relevant when putting ML systems in production is that of fairness or more generally ethical aspects of ML predictions [40]. There are a number of examples when ML technology used as assistive technology for instance in border control[5] or policing[6] led to racist or otherwise biased decisions. It has been widely recognized in the ML research community that when building ML applications validating the ML predictions with respect to fairness is a fundamental prerequisite. Evaluating this dimension is however amongst the most difficult of all validation challenges: there are different fairness definitions, and even if one could agree on a small set of definitions, validating these requires insight into the respective grouping variables, which are, for ethical and legal reasons, typically not available. The relevant grouping information is thus often reflected in a data set through other features, which are considered not ethically problematic, such as zip code, but which are often correlated with ethically sensitive features. Other implicit biases of data sets can emerge from data preprocessing, for instance how missing values are dealt with when they are distributed not equally across ethnical groups [65]. Detecting these biases requires not only domain expertise, but also often information that is not contained in the data set itself.

3 Current Solutions

Validating data at each of the stages in a ML workflow as sketched in Figure 1 requires a wide range of competencies from data engineering over ML algorithms to user interface design. These competencies are often distributed across different roles in a team, and especially in small (5 to 10 members) teams, it is not unlikely that only one person has the competency to dive deep into a root cause analysis of the various data validation dimensions sketched in section 2. Solving problems in an on-call situation or when scaling up an ML system requires reliable automation for the data validation. As many of these validations have to be data dependent, such automation often amounts to *using ML to validate data in ML workflows* which has, for instance, been done for outlier removal [68], data cleaning [34] or missing value imputation [8].

[5]`https://www.bbc.com/news/technology-53650758`
[6]`https://www.nytimes.com/2019/07/10/opinion/facial-recognition-race.html`

There are, however, simpler approaches that do not leverage ML models to validate data in ML systems, such as Tensorflow Extended (TFX) [6] or DEEQU [54]. In the following we will highlight some of the state of the art solutions to automated data validation in the context of ML systems.

3.1 Schema Validation

Validating syntactic correctness, also referred to as schema validation, is – at first glance – relatively simple and often integral part of data ingestion. Usually data violating data type constraints will immediately raise errors and break a data pipeline. Yet, these validations are often counteracted by humans. In practice it is not uncommon to use generic data types such as strings for data that is actually numeric, ordinal or categorical. While this will ensure that schema violations are minimized, as any data type can be cast to string, it imposes additional work on any downstream validation for data correctness and consistency. This problem is worsened by the fact that some software libraries popular in the ML community often cast string data that contains numeric data automatically to numeric data types. This can have effects on entire scientific communities. A recent study reports that casting errors induced by excel in biomedical data occur in approximately one fifth of the papers in leading genomics journals [70].

While simple validation such the data type can be trivially automated, it is unlikely that this will prevent engineers from having to deal with data type errors. There will always be on-call engineers who need to fix problems as fast as possible and the cost of having stale or broken data pipelines in a production system is often far bigger then the difficulty to quantify cost of accumulating hidden technical debt by specifying generic data types like strings and thereby circumventing data type checks. This is why this simple task of data type detection is actually an interesting research challenge. Often simple heuristics will do the job and the popular libraries use these heuristics to determine the data type of columns. The simplest heuristic is probably to try and cast a value to a certain data type and try other data types in case an error is raised[7]. Such heuristics often work well in practice but can be insufficient, especially when dealing with heterogeneous data, such as mixed data types in one column. Also, for the seemingly simple task of automating data type inference, there is interesting recent work on using ML models to infer a probabilistic notion of the most likely data type given the symbols used in a data base [12]. Approaches combining scalable heuristics with more sophisticated ML based data type inference techniques are a promising alternative to the current situation in which restrictive data type checks and broken data pipelines often lead data engineers to opt for too generic data types.

3.2 Data Correctness and Data Consistency

Validation of data consistency and correctness are one of the areas where classical DBMS research has made significant contributions, for instance in the data profiling literature [3]. Popular approaches to validate correctness and consistency are rule based systems such as *NADEEF* [13]. There are commercial versions of simple rule based systems, a popular example is trifacta[8]. And there are open source systems for semi-automated validation of consistency, such as Open-Refine[9]. Other approaches primarily investigated in an academic context include attempts to learn *(approximate/relaxed) functional dependencies* between columns of a table [46]. These functional dependencies between columns include for example a dependency between a column containing zip codes and another column containing city names. These dependencies can be used to automatically learn validation checks related to consistency and correctness [50]. Such approaches are well established and there exist efficient methods for large scale functional dependency discovery [35]. Yet, in the context of data pipelines for ML systems none of these approaches have reached broad adoption. We discuss some potential reasons in section 4.

[7]https://github.com/pandas-dev/pandas/blob/v1.1.4/pandas/core/dtypes/common.py#L138
[8]https://www.trifacta.com/
[9]https://openrefine.org/

In addition to the solutions of the DBMS community there are a number of approaches developed at the intersection of DBMS and ML communities, tailored to ML specific workflows [11, 54]. The research is accompanied by open source libraries[10][11] that implement schema validation by checking for correct data types, but they also offer validation options for or other basic data properties, for instance whether the values are contained in a predefined set. Prominent examples for a data validation solution for ML systems are the ones integrated in SageMaker Data Wrangler[12] and TFX [11]. Their schema validation conveniently allows users to define integrity checks on the serving data in an ML system, e.g. for data type or value ranges. Schema validation parameters can also be derived from the training data, which allows to automate many of these checks.

3.3 Validation of Statistical Properties

All data sets are subject to noise, be it because humans enter wrong values into a form or because human software developers write code that leads to faulty data ingestion. Dealing with this noise is difficult for exact validation rules such as functional dependencies. Statistical learning techniques can help to deal with noisy data.

Error-Detection Solutions Taking a probabilistic stance on DBMSs is not a new idea [48]. Recently, the idea of using statistical techniques for data validation tasks has become a major focus of research. One example are statistical learning approaches to error detection in data bases [50, 25, 29, 39]. These approaches have the potential to automatically validate data correctness and to automatically clean data sets. Some of these systems can be automated to a large extent [32, 34], others rely on a semi-supervised approach [33].

ML Frameworks for Statistical Data Validation Also data validation solutions like TFX data validation [11] or DEEQU [54] address data validation including statistical rules, such as deviation of values around the mean of a numeric column. These validation solutions can be very helpful, if one knows what to test for. But data sources can easily have billions of rows and hundreds of columns. For these cases it can infeasible to manually create, validate and maintain data quality checks. To alleviate the burden of manually creating constraints, the authors of DEEQU [54] propose to utilize historic data to generate column profiles and generate data quality constraints from these. These quality constraints can also make use of the temporal structure of data quality metrics collected over time using time series models or other anomaly detection methods.

Anomaly Detection Methods Instead of applying them to metrics computed on a data set, anomaly detection methods are also often used to detect anomalies in the data tuples directly. There are many methods available in easy to use software libraries, see for instance [68], and there are commercial products that allow to automate anomaly detection in large scale industrial settings[13]. While the goal of these anomaly detection approaches is similar to the above error detection approaches originating in the DBMS community, many anomaly detection solutions emerged from the ML community. One of the most important differences is that anomaly detection approaches, as most ML methods, usually expect the data to come in matrix form, with all columns being numeric. Most methods from the DBMS community expect the data to be high cardinality categorical data, some also are applicable to numeric values, but that is not very common in research on functional dependencies for instance [46]. So applying anomaly detection methods to heterogeneous data sets with non-numeric (categorical, ordinal or even free text data) requires to apply *feature extraction* techniques to bring the data into a numeric format. This is a standard preprocessing step in ML pipelines, but popular software libraries for anomaly detection, such as [68], do not include this important preprocessing step. This makes these libraries difficult to apply for automated

[10]https://www.tensorflow.org/tfx/guide/tfdv

[11]https://github.com/awslabs/deequ

[12]https://aws.amazon.com/sagemaker/data-wrangler/

[13]https://aws.amazon.com/de/blogs/big-data/real-time-clickstream-anomaly-detection-with-amazon-kinesis-analytics/

data validation. It is possible to integrate standard anomaly detection methods as statistical data validation steps in ML systems, but this imposes two additional challenges onto the engineering team. For one, this integration requires to write 'glue code' for the feature extraction – which is often one of the major sources for accumulating *technical debt* in a software system. And secondly this requires to have a good evaluation metric for the anomaly detection. Which is, in contrast to standard supervised learning scenarios, often difficult to define and get ground truth data for.

Model Robustness and Generalization Performance Another central problem with all of the above approaches to statistical data validation in a ML context is that most of these methods are blind to the impact of data errors on downstream ML components. This is however one of the most important aspects for ML systems. Often it does not matter for the ML model, whether a feature is affected by a data shift, for instance when regularization of an ML model forces the ML model to ignore a feature. And in other cases tiny changes in the input, which human observers would not be able to detect, can have devastating consequences on the ML model output [4]. Recent work in the ML community has shown that especially novel deep learning methods can suffer from severe stability problems [14]. Some aspects of this can be mitigated by employing standard ML concepts such as k-fold cross-validation (CV). This approach has unfortunately lost popularity due to the sheer compute demand of modern deep learning methods. Most deep learning papers usually use just a single train/validation/test split. Standard nested k-fold CV can have decisive advantages when it comes to measuring robustness of a ML model. However, these techniques only work when there is ground truth data available. In a production setting, this is often not the case. There exist however also other methods to measure the robustness of ML models when there is no ground truth data available. For instance in [55] the authors leverage a set of declaratively defined data errors applied to data for which ground truth is available and measure the predictive performance of a ML model under these perturbations. This allows to train a meta model that can be used to predict the predictive performance on new unseen data with high precision. Such an approach can be useful in production ML systems to automatically validate data errors with respect to their impact on downstream ML model performance.

3.4 Fairness Validation

Fairness is one of the most prominent examples of how ML systems can fail and severely influence the public opinion about a company or an entire scientific community. It is thus of utmost importance to ensure that this dimension of data validation in the context of ML systems is not neglected. Yet validating this dimension is especially difficult, for a number of reasons. First and foremost it is difficult to define fairness. An excellent overview over the last decades of fairness research with a focus on ML systems can be found in [40]. The main insight here is that fairness validation it is not only a technical challenge. Instead, it is imperative to include multiple scientific disciplines in this research, in particular also researchers from sociology, psychology and law. Setting the stage for such transdisciplinary research is a challenge in itself, for instance finding a common terminology is not trivial. But we have seen that the research community has made progress by fostering transdisciplinary discussions at newly emerging conferences[14]. The joint efforts of different communities have helped to identify many ways in which biases leading to unfair ML systems can enter a workflow. Many of these biases arise in the data generating process. Enabling scientists and engineers to identify such biases should be part of the research agenda of the data management and ML community [60]. One example in this direction is *FairBench* [65], an open source library that helps to trace changes in data distributions and visualize distortions with respect to protected group memberships throughout the pipeline. Another example is SageMaker Clarify[15], an explainability feature for Amazon SageMaker that provides insights into data and ML models by identifying biases and explaining predictions. It is deeply integrated into Amazon SageMaker, a fully managed service

[14]See for instance `https://dl.acm.org/conference/fat`

[15]https://aws.amazon.com/sagemaker/clarify/

that enables data scientists and developers to build, train, and deploy ML models at any scale. Clarify supports bias detection and feature importance computation across the ML lifecycle, during data preparation, model evaluation, and post-deployment monitoring. Libraries like these are a prerequisite to better automation of fairness validation. However, another more fundamental problem of fairness validation remains even if technical solutions for visualizing and detecting biases are available: Group membership variables are required for most technical solutions to fairness. Storing these variables can be more challenging, from an operational stance, than storing other non-confidential data.

3.5 Privacy Validation

Similar to fairness, also privacy is a difficult to define validation dimension. However in contrast to fairness, the research community has converged to a few concepts that are relatively straightforward in terms of their definitions. Popular concepts include *differential privacy* [17] and *k-anonymity* [61], see also section 2. Most ML related work on privacy focuses on differential privacy, where noise is added the data, gradients or the model coefficients. Validating and trading off privacy against predictive performance or *utility* of the ML model can be challenging [30]. Empirically, evaluating privacy is often done using *membership inference attacks* [59], which has also been adopted for unsupervised ML models [24]. One limitation of these approaches is that privacy validation is always dependent on a specific model and data set. General statements about privacy and utility independent of models and data is hence difficult.

3.6 Validation of Robustness against adversarial attacks

Privacy validation is aiming at defending the ML system against a certain type of adversarial attack, where for instance the identity of data points used for training the ML system is revealed. There are however other types of adversarial attacks, for instance when artificial examples are generated to result in ML predictions with a certain outcome. Validation of robustness against such types of attacks can be achieved by perturbations around the data manifold [45, 38]. This can be achieved by extracting latent representations of the input data [26] or of the predictions [7, 19]. Alternative methods rely on training an additional classifier used to decide whether an example is adversarial or not [21, 23, 41]. Complementary to the work on validating adversarial robustness, a lot of work has been devoted to making ML models more robust to adversarial attacks by augmenting the training datasets with adversarial examples [22, 38, 66, 69].

3.7 Human in the loop evaluation

Most of the above data quality dimensions are easy for humans to assess. This is why human audits are still one of the most direct and most robust options for data validation in ML systems. Expert auditors, such as researchers developing a new service, often can quickly identify errors and their root causes by simply inspecting input and outputs of a ML system. Among the most important disadvantages with this approach is that these validations are expensive and do not scale well. Sometimes human-in-the-loop validations can be scaled up using crowd-sourcing platforms such as Yandex' Toloka or Amazon Mechanical Turk. Increasing the quality of crowd-sourced validations is an active topic of ML research [64]. For instance there are attempts to render audits more efficient by including transparency of ML model predictions [56] or by providing more inciting incentives [27, 63]. Still, this approach can be difficult to automate and is generally used as an andon cord in a modelling pipeline rather than an integrated quality test. This is not only due to the fact that it is difficult to integrate human audits in build processes. Focussing on human judgements only can lead to biased validations, especially when using transparent ML methods [57].

4 Conclusion

Validation of input and output data in ML production systems has many facets that require competencies often distributed across a heterogeneous team of engineers and scientists, as illustrated in Figure 1. While some of the data validation challenges, such as schema validation or data consistency, can be tackled with traditional data profiling methods established the DBMS community, other validation dimensions are specific to ML systems. These ML model dependent validation challenges include aspects like accuracy and robustness under data shifts, fairness of ML model predictions and privacy concerns.

In section 3 we highlight a number of solutions to validate single aspects. Many of these approaches are typically tailored to specific use cases and often require considerable integration efforts in production ML systems. A good example are the various solutions from both the ML community as well as the DMBS community for checking simple data properties, such as the data types, and also more complex dimensions like data consistency. Many of these approached allow for automating the generation of validation checks. Yet in practice it is not trivial to automate the generation of validations for a new ML system that ingests and produces millions of rows and columns. For instance, there are many cases when simple validation checks on input data will lead to false alarms when shifts or errors in a single feature do not impact the output of a downstream ML model – maybe because that feature was neglected by the ML model, when strong regularization during the ML model training phase taught the model to ignore that feature.

Despite the rapid progress in recent years to automate and monitor ML systems: to the best of our knowledge there exists no data validation system that has reached broad adoption and which takes into account all of the data validation dimensions sketched in section 2. One reason for this is the difficulty of combining the multitude of validation strategies listed in section 3 into one unified framework. Considering the rapid pace of research at the intersection of ML and DBMS, see for instance [15], it is fair to assume that it is merely a matter of a few years until one framework or some open standard for data validation in the context of ML systems will have reached broad adoption.

There are many data validation challenges in ML systems that go beyond technical aspects. Many of them are due to the complex nature of the data transformations induced by ML models. For instance identifying unfair biases often requires domain knowledge or access to grouping variables, which are often not available. And even if those are available, it is not always clear how fairness in the context of ML systems can be defined [67]. A conceptual challenge related to privacy is for instance the trade-off between utility and differential privacy of a ML system [30]: how much predictive accuracy should be sacrificed to ensure privacy? Sacrificing accuracy against privacy in domains like health care or jurisdiction is a difficult question for which ethical and legal dimensions are more important than technical aspects. Next to these ethical and legal aspects, there is one key factor hindering adoption of comprehensive data validation in ML systems and that more related to cultural aspects. Many scientists like to build new models and tools, but writing tests, integrating monitoring and validation stages in an ML system are not exactly the most popular tasks amongst researchers. But often the competencies of the scientists who built a model is required to build well functioning monitoring and validation solutions in ML systems.

Based on these observations we derive some suggestions for how to drive innovation and adoption of data validation in the context of ML systems. First, we hope that the current trend for research at the intersection of ML and DBMS communities will continue to grow and identify more synergies leveraging and complementing each others expertise. We have seen some great examples of constructive but vivid discussion between the two communities, for instance that sparked by Kraska and colleagues around their work on learning index structures [31]. This work is unrelated to data validation and mentioned merely as an example of transdisciplinary research debates. Second, when building ML systems there is a broad spectrum of operational challenges and seamless integration with cloud infrastructure is key to reaching broad adoption.

We conclude that establishing data validation in ML systems will require a stronger focus on usability and simple APIs. Third we believe that data validation in ML systems will reach broad adoption once the research community will have found better ways of automating the validation workflow, most importantly the generation of checks for each of the data validation dimensions listed in section 2.

In the past years we have seen great examples of automated tooling for tracking ML experiments [53], experimentation in ML production systems [10], input data validation [54, 11] and validation strategies for predictions of ML systems [49, 55, 14]. One example of how some of these data validation techniques could be integrated into an automated workflow would be that presented in [51], where the authors propose to iterate through a sequence of data validation [54], data cleaning[8] and quantification of downstream impact on ML predictive performance [55] to achieve an automated ML workflow. We believe that increasing the level of usability through automation in data validation will enable researchers to focus on more important questions like the conceptual, ethical and legal questions and ultimately lead to more responsible usage of ML systems in production systems.

References

[1] *Data Lifecycle Challenges in Production Machine Learning: A Survey*, number (Vol. 47, No. 2). SIGMOD Record, 2018.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. Data Profiling. *Synthesis Lectures on Data Management*, 10(4):1–154, nov 2018.

[4] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. volume 80 of *Proceedings of Machine Learning Research*, pages 284–293, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[5] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. Differential privacy has disparate impact on model accuracy. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 15479–15488. Curran Associates, Inc., 2019.

[6] Denis Baylor, Eric Breck, Heng Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. TFX: A TensorFlow-based production-scale machine learning platform. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume Part F129685, pages 1387–1395, New York, NY, USA, aug 2017. Association for Computing Machinery.

[7] Arjun Nitin Bhagoji, Daniel Cullina, Chawin Sitawarin, and Prateek Mittal. Enhancing robustness of machine learning systems via data transformations. *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, Mar 2018.

[8] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. "Deep" Learning for Missing Value Imputationin Tables with Non-Numerical Data. In *Proceedings of the 27th ACM*

International Conference on Information and Knowledge Management - CIKM '18, pages 2017–2025, New York, New York, USA, 2018. ACM Press.

[9] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. *Lecture Notes in Computer Science*, pages 387–402, 2013.

[10] Joos Hendrik Böse, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias Seeger, and Yuyang Wang. Probabilistic demand forecasting at scale. *Proceedings of the VLDB Endowment*, 10(12):1694–1705, 2017.

[11] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. DATA VALIDATION FOR MACHINE LEARNING. Technical report, 2019.

[12] Taha Ceritli, Christopher K.I. Williams, and James Geddes. ptype: probabilistic type inference. *Data Mining and Knowledge Discovery*, 34(3):870–904, may 2020.

[13] Michele Dallachiesat, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. NADEEF: A commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2013.

[14] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. Underspecification Presents Challenges for Credibility in Modern Machine Learning. 2020.

[15] Xin Luna Dong and Theodoros Rekatsinas. Data Integration and Machine Learning: A Natural Synergy. *Proceedings ofthe VLDB Endowment, Vol.*, 11(12):2094–2097, 2018.

[16] Cynthia Dwork. Differential privacy: A survey of results. In *In Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[17] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[19] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. Detecting adversarial samples from artifacts, 2017.

[20] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2015.

[21] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins, 2017.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[23] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples, 2017.

[24] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. LOGAN: Membership Inference Attacks Against Generative Models. *Proceedings on Privacy Enhancing Technologies*, 2019(1):133–152, 2019.

[25] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. HoloDetect: Few-Shot Learning for Error Detection. *ACM Reference Format*, 2019.

[26] Dan Hendrycks and Kevin Gimpel. Early methods for detecting adversarial images, 2016.

[27] Chien-Ju Ho, Aleksandrs Slivkins, Siddharth Suri, and Jennifer Wortman Vaughan. Incentivizing high quality crowdwork. *Proceedings of the 24th International Conference on World Wide Web - WWW '15*, 2015.

[28] H. Hosseini, S. Kannan, and R. Poovendran. Are odds really odd? bypassing statistical detection of adversarial examples. *ArXiv*, abs/1907.12138, 2019.

[29] Zhipeng Huang and Yeye He. Auto-Detect: Data-Driven Error Detection in Tables. *SIGMOD*, 2017.

[30] Bargav Jayaraman and David Evans. *Evaluating Differentially Private Machine Learning in Practice*. 28th USENIX Security Symposium, 2019.

[31] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 489–504, 2018.

[32] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. BoostClean: Automated Error Detection and Repair for Machine Learning.

[33] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and K. Goldberg. ActiveClean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016.

[34] Sanjay Krishnan and Eugene Wu. AlphaClean: Automatic Generation of Data Cleaning Pipelines. apr 2019.

[35] Sebastian Kruse and Felix Naumann. Efficient Discovery of Approximate Dependencies. *PVLDB*, 11(7):759–772, 2018.

[36] Arun Kumar, Matthias Boehm, and Jun Yang. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*, 2017.

[37] A. Kurakin, Ian J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *ArXiv*, abs/1607.02533, 2017.

[38] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[39] Mohammad TU Mahdavi Berlin mahdavilahijani, tu-berlinde Ziawasch Abedjan, Raul Castro Fernandez MIT, Samuel Madden MIT, Mourad Ouzzani, Michael Stonebraker MIT, Nan Tang, Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, and Michael Stonebraker. Raha: A Configuration-Free Error Detection System. 18(19).

[40] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A Survey on Bias and Fairness in Machine Learning. aug 2019.

[41] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations, 2017.

[42] I. Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[44] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[45] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples, 2017.

[46] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, jun 2015.

[47] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data, 2016.

[48] Michael Pittarelli. Roger Cavallo. *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, (January):71–81, 1987.

[49] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: An empirical study of methods for detecting dataset shift, 2018.

[50] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings ofthe VLDB Endowment*, 10(11), 2017.

[51] Tammo Rukat, Dustin Lange, Sebastian Schelter, and Felix Biessmann. Towards Automated ML Model Monitoring: Measure, Improve and Quantify Data Quality. Technical report.

[52] Wojciech Samek and Klaus Robert Müller. Towards Explainable Artificial Intelligence. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11700 LNCS, pages 5–22. Springer Verlag, 2019.

[53] Sebastian Schelter, Joos-Hendrik Boese, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. Automatically tracking metadata and provenance of machine learning experiments. *Machine Learning Systems workshop at NeurIPS*, 2017.

[54] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, Andreas Grafberger, and Meltem Ce-Likel. Automating Large-Scale Data Quality Verification. *PVLDB*, 11(12):1781–1794, 2018.

[55] Sebastian Schelter, Rukat Tammo, and Felix Biessmann. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. *SIGMOD*, 2020.

[56] Philipp Schmidt and Felix Biessmann. Quantifying interpretability and trust in machine learning systems, 2019.

[57] Philipp Schmidt and Felix Biessmann. Calibrating human-ai collaboration: Impact of risk, ambiguity and transparency on algorithmic bias. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 431–449, Cham, 2020. Springer International Publishing.

[58] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. *Nips*, pages 2494–2502, 2015.

[59] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. *Proceedings - IEEE Symposium on Security and Privacy*, pages 3–18, 2017.

[60] Julia Stoyanovich, Bill Howe, and H. V. Jagadish. Responsible data management. *Proceedings of the VLDB Endowment*, 13(12):3474–3488, 2020.

[61] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[63] H. Wang, S. Guo, J. Cao, and M. Guo. Melody: A long-term dynamic quality-aware incentive mechanism for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):901–914, 2018.

[64] Jennifer Wortman Vaughan. Making Better Use of the Crowd: How Crowdsourcing Can Advance Machine Learning Research. Technical report, 2018.

[65] Ke Yang, Biao Huang, and Sebastian Schelter. Fairness-Aware Instrumentation of Preprocessing Pipelines for Machine Learning. 2020.

[66] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training, 2019.

[67] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering*, pages 1–1, feb 2020.

[68] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 2019.

[69] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, pages 1399–1407, New York, NY, USA, 2019. Association for Computing Machinery.

[70] Mark Ziemann, Yotam Eren, and Assam El-Osta. Gene name errors are widespread in the scientific literature. *Genome Biology*, 17(1):177, aug 2016.

Enhancing the Interactivity of Dataframe Queries by Leveraging Think Time

Doris Xin, Devin Petersohn, Dixin Tang, Yifan Wu, Joseph E. Gonzalez,
Joseph M. Hellerstein, Anthony D. Joseph, Aditya G. Parameswaran
UC Berkeley

1 Introduction

During the course of Machine Learning (ML) model development, a critical first step is *data validation*, ensuring that the data meets acceptable standards necessary for input into ML training procedures. Data validation involves various sub-tasks, including *data preparation*: transforming the data into a structured form suitable for the desired end-goal, and *data cleaning:* inspecting and fixing potential sources of errors. These validation steps of data preparation and cleaning are essential even if the eventual goal is simply exploratory data analysis as opposed to ML model development—in both cases, the quality of the eventual end-result, be it models or insights, are highly tied to these steps. This data validation process is highly exploratory and iterative, as the data scientist often starts off with a limited understanding of the data content and quality. Data scientists therefore perform data validation through *incremental trial-and-error*, with the goals evolving over time: they make a change, inspect the result (often just a sample) to see if it has improved or "enriched" the dataset in some way, e.g., by removing outliers or filling in NULL values, expanding out a nested representation to a flat relational one, or pivoting to organize the dataset in a different manner more aligned with the analysis goals.

To support this iterative process of trial-and-error, data scientists often use powerful data analysis libraries such as Pandas [7] within computational notebooks, such as Jupyter or Google Colab [12, 1]. Pandas supports a rich set of incrementally specified operators atop a tolerant dataframe-based data model, drawn from relational algebra, linear algebra, and spreadsheets [14] embedded within a traditional imperative programming language, Python. While the use of dataframe libraries on computational notebooks is a powerful solution for data validation on small datasets, this approach starts to break down on larger datasets [14], with many operations requiring users to wait for unacceptably long periods, breaking flow. Currently, this challenge may be overcome by either switching to a *distributed dataframe system* (such as Dask [3] and Modin [6]), which introduces setup overhead and potential incompatibilities with the user's current workflow, or by users manually optimizing their queries, which is a daunting task as pandas has over 200 dataframe operations. We identify two key opportunities for improving the interactive user experience *without requiring changes to user behavior*:

- Users often do not want to inspect the entire results of every single step.
- Users spend time thinking about what action to perform next.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Unfortunately, at present, every cell (the unit of execution in a notebook) issued by the user is executed verbatim immediately, with the user waiting until execution is complete to begin their next step. Moreover, the system is idle during *think time*, i.e., when users are thinking about their next step or writing code. Fundamentally, *specification* (how the user writes the query) and *execution* (what the system executes) are tightly coupled.

In this paper, we outline our initial insights and results towards optimizing dataframe queries for interactive workloads by *decoupling specification and execution*. In particular, dataframe queries are not executed immediately, unless the user intends to inspect the results, but are deferred to be computed during think time. We distinguish operators that produce results that users inspect, what we call *interactions*, from those that do not. We can then use program slicing to quickly determine what code is *critical* in that it influences the interactions, i.e., what the user intends to see immediately, and what is *non-critical*, in that it can be computed in the background during think-time to speed up future interactions. For the critical portions, we further identify if it can be rewritten in ways that allows us to improve interactivity further. For example, identifying that users often only examine the first or last few rows/columns of the result allows us to compute this as part of the critical portion and defer the rest to the non-critical portion. For the non-critical portions, by deferring the execution of the non-critical portions, we can perform more holistic query planning and optimization. Moreover, we may also *speculatively* compute other results that may prove useful in subsequent processing. We call our framework *opportunistic evaluation*, preserving the benefits of eager evaluation (in that critical portions are prioritized), and lazy or deferred evaluation (in that non-critical portions are deferred for later computation). This paper builds on our prior vision [14], wherein we outline our first steps towards establishing a formal framework for reasoning about dataframe optimization systematically.

2 Background and Motivation

2.1 Key Concepts

Users author dataframe queries in Jupyter notebooks, comprising code cells and output from executing these code cells. Figure 1 shows an example notebook containing dataframe queries on the left. Each code cell contains one or more queries and sometimes ends with a query that outputs results. In this part of Figure 1, every cell ends in a query (namely, `df1.describe()`, `df1.head()`, and `df2.describe()`) that outputs results. Dataframe queries are comprised of operators such as `apply` (applying a user defined function on rows/columns), `describe` (compute and show summary statistics), and `head` (retrieve the top K rows of the dataframe). Operators such as `head` and `describe`, or simply the dataframe variable itself, are used for inspecting intermediate results. We call these operators ***interactions***. Users construct queries incrementally by introducing interactions to verify intermediate results. An interaction usually depends on only a subset of the operators specified before it. For example, `df1.describe()` in Figure 1 depends only on `df1 = pd.read_csv("small_file")` but not `df2 = pd.read_csv("LARGE_FILE")`. We call the set of dependencies of an interaction the ***interaction critical path***. To show the results of a particular interaction, the operators not on its interaction critical path do not need to be executed even if they were specified before the interaction.

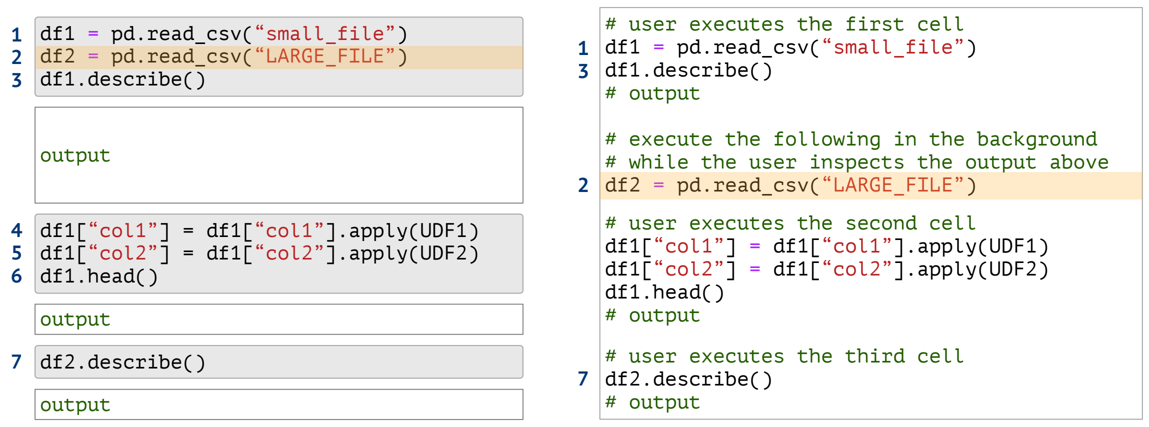
After an interaction, users spend time inspecting the output and authoring new queries based on the output. We call the time between the display of the output and the submission of the next query ***think time***, during which the CPU is idle (assuming there are no other processes running on the same server) while the user inspects intermediate results and authors new queries. We propose ***opportunistic evaluation***, an optimization framework that leverages this think time to reduce interactive latency. In this framework, the execution of operators that are not on interaction critical paths, which we call *non-critical operators*, are deferred to being evaluated asynchronously during think time to speed up future interactions.

2.2 Motivating Scenarios and Example Optimizations

To better illustrate the optimization potential of opportunistic evaluation, we present two typical data analysis scenarios that could benefit from asynchronous execution of queries during think time to minimize interactive latency. While the user's program remains the same, we illustrate the modifications to the execution plan that highlights the transformations made.

2.2.1 Interaction-based Reordering

Consider a common workflow of analyzing multiple data files, shown on the left in Figure 1. The user, Sam, executes the first cell, which loads both of the files, and is forced to wait for *both* to finish loading before she can interact with *either* of the dataframes. To reduce the interactive latency (as perceived by the user), we could conceptually re-order the code to optimize for the immediate output. As shown on the right in Figure 1, the re-ordered program defers loading the large file to after the interaction, `df1.describe()`, obviating the need to wait for the large file to load into `df2` before Sam can start inspecting the content of the small file. To further reduce the interactive latency, the system could load `df2` while Sam is viewing the results of `df1.describe()`. This way, the time-consuming process of loading the large file is completed during Sam's think time, thus reducing the latency for interacting with `df2`.



```
1 df1 = pd.read_csv(“small_file”)
2 df2 = pd.read_csv(“LARGE_FILE”)
3 df1.describe()

  output

4 df1[“col1”] = df1[“col1”].apply(UDF1)
5 df1[“col2”] = df1[“col2”].apply(UDF2)
6 df1.head()

  output

7 df2.describe()

  output
```

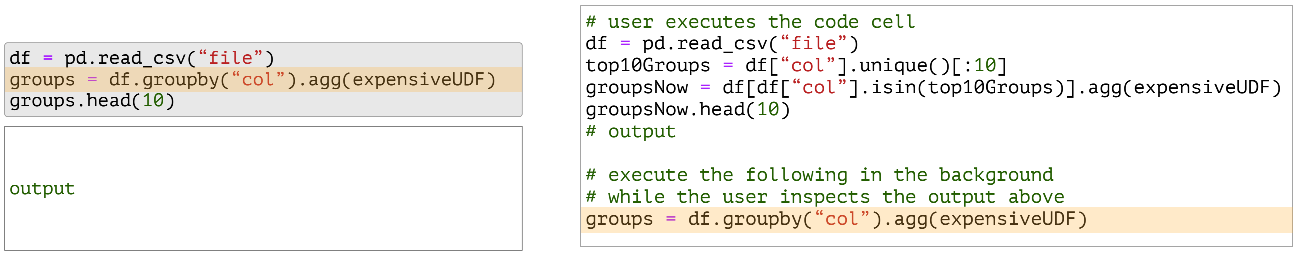
```
  # user executes the first cell
1 df1 = pd.read_csv(“small_file”)
3 df1.describe()
  # output

  # execute the following in the background
  # while the user inspects the output above
2 df2 = pd.read_csv(“LARGE_FILE”)

  # user executes the second cell
  df1[“col1”] = df1[“col1”].apply(UDF1)
  df1[“col2”] = df1[“col2”].apply(UDF2)
  df1.head()
  # output

  # user executes the third cell
7 df2.describe()
  # output
```

Figure 1: Example program transformation involving operator reordering: (left) Original program where user has to wait for both files to load before viewing any; (right) Optimized program where the user can view the smaller file first while the other loads.

2.2.2 Prioritizing Partial Results

For any large dataframes, users can only inspect a handful of rows at a time. However the current evaluation mechanism requires *all* the rows to be evaluated. Expensive queries such as those involving user-defined functions (UDFs) could take a long time to fully compute, as shown on the left in Figure 2.

To reduce interactive latency, one can prioritize computation of only the portion of the dataframe inspected. This method is essentially an application of *predicate pushdown*, a standard technique from database query optimization. The right part of Figure 2 provides an example transformation for the particular operator, `groupby`. While the first cell prioritizes the computation of the inspected rows, the user may still need the result of the entire computation, which is scheduled to be computed later while Sam is still reading the result of the previous cell, `groupNow.head(10)`, i.e. the think time. A noteworthy attribute of dataframes is row and column equivalence [14], which means that predicate pushdown can also happen when projecting columns as well.



```
df = pd.read_csv("file")
groups = df.groupby("col").agg(expensiveUDF)
groups.head(10)
```

```
output
```

```
# user executes the code cell
df = pd.read_csv("file")
top10Groups = df["col"].unique()[:10]
groupsNow = df[df["col"].isin(top10Groups)].agg(expensiveUDF)
groupsNow.head(10)
# output

# execute the following in the background
# while the user inspects the output above
groups = df.groupby("col").agg(expensiveUDF)
```

Figure 2: Program transformation involving predicate pushdown. (left) Original program where the user has to wait for an expensive UDFs to fully compute; (right) Optimized program where the user can view a partial result sooner.

3 Assessment of Opportunities with Notebook Execution Traces

To assess the size of opportunity for our aforementioned optimizations to reduce interactive latency in computational notebooks, we evaluate two real world notebook corpora.

One corpus is collected from students in the **Data 100** class offered at UC Berkeley. Data 100 is an intermediate data science course offered at the undergraduate level, covering topics on tools and methods for data analysis and machine learning. This corpus contains 210 notebooks across four different assignments, complete with the *history* of cell execution content and completion times captured by instrumenting a custom Jupyter extension.

We also collected Jupyter notebooks from **Github** comprising a more diverse group of users than Data 100. Jupyter's IPython kernel stores the code corresponding to each individual cell executions in a local `history.sqlite` file[1]. We used 429 notebook execution histories that Macke et al. [13] scraped from Github that also contained pandas operations.

To assess optimization opportunities, we first quantify think time between cell executions, and then evaluate the prevalence of the code patterns discussed in Section 2.2.

3.1 Think-Time Opportunities

Our proposed opportunistic evaluation framework takes advantage of user think time to asynchronously process non-critical operators to reduce the latency of future interactions. To quantify think time, we measure the time lapsed between the completion of a cell execution and the start of the next cell execution using the timestamps in the cell execution and completion records, as collected by our Jupyter notebook extension. Note that the think time statistics are collected only on the Data 100 corpus, as the timestamp information is not available in the Github corpus. Figure 3 shows the distribution of think time intervals on the left, measured in seconds, between consecutive cell executions across all notebooks, while the right part of Figure 3 shows the distribution of the median think time intervals, measured in seconds, within each notebook. We removed automatic cell re-execution ("run all") from the dataset. We can see that while there are many cells that were executed quickly, there exist cells that had ample think time—the 75th percentile think time is 23 seconds.

3.2 Program Transformation Opportunities

Interaction-Based Reordering. To assess the opportunities to apply operator reordering to prioritize interactions, we evaluate the number of non-critical operators specified before each interaction. We use the operator DAG, to be described in Section 4.2, to determine the dependencies of an interaction and count the number of operators that

[1]https://ipython.readthedocs.io/en/stable/api/generated/IPython.core.history.html

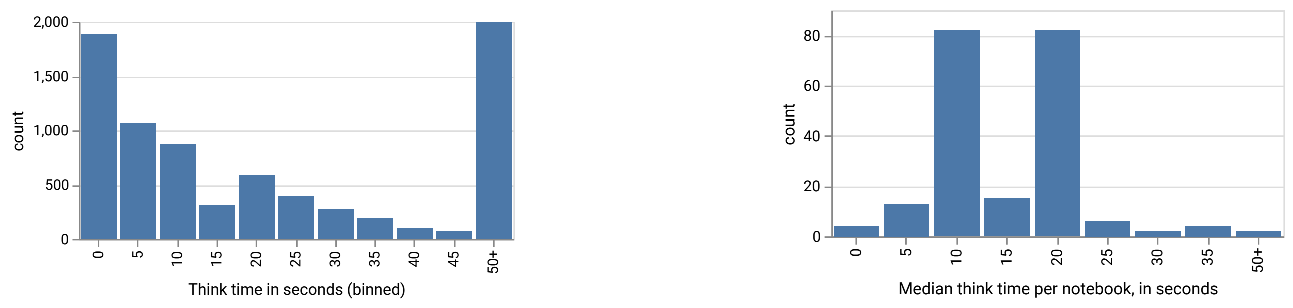


Figure 3: Think time the average "think time" between cell executions and the average think time per notebook. (left) Think time between cell executions; (right) Median think time per notebook across cells.

are not dependencies, i.e., non-critical operators, specified above the interaction. Figure 4 shows the distributions for the two datasets. In both cases, non-critical operators present a major opportunity: the Data 100 and Github corpus have, respectively, 54% and 42% interactions with non-critical operators.

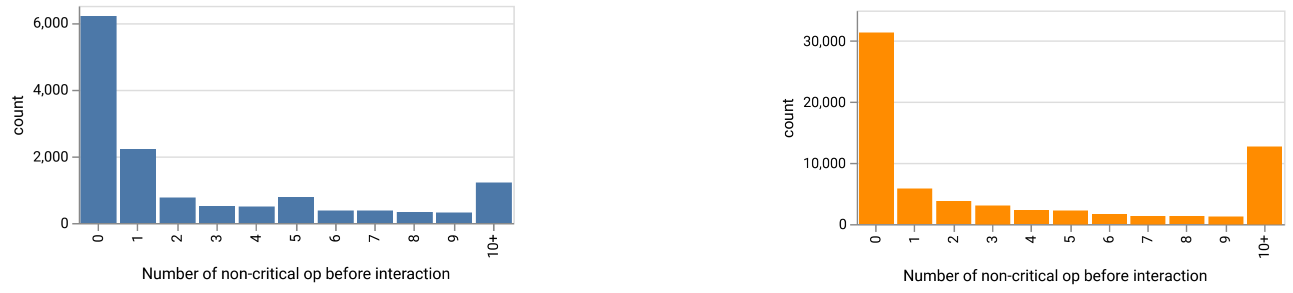


Figure 4: Number of non-critical operators before interactions. (left) Data 100: $\mu = 4$, $\sigma = 5$; (right) Github: $\mu = 7$, $\sigma = 11$

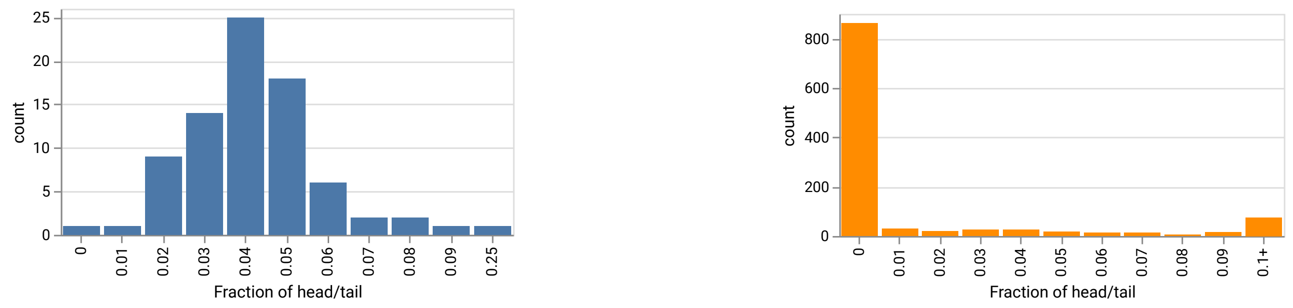


Figure 5: Stats for head/tail interactions used in each notebook. (left) Data 100: $\mu = 0.04$, $\sigma = 0.028$; (right) Github: $\mu = 0.11$, $\sigma = 0.21$

Prioritizing Partial Results. The optimization for prioritizing partial results via predicate pushdown can be applied effectively to many cases when predicates are involved in queries with multiple operators. The most common predicates in the dataframe setting are `head()` and `tail()`, which show the top and bottom K rows of the dataframe, respectively. Figure 5 show the distribution of the fraction of interactions that are either `head` or `tail` in each notebook. We see that partial results views are much more common in the GitHub dataset than Data 100. This could be due to the fact that users on GitHub tend to keep the cell output area short for better

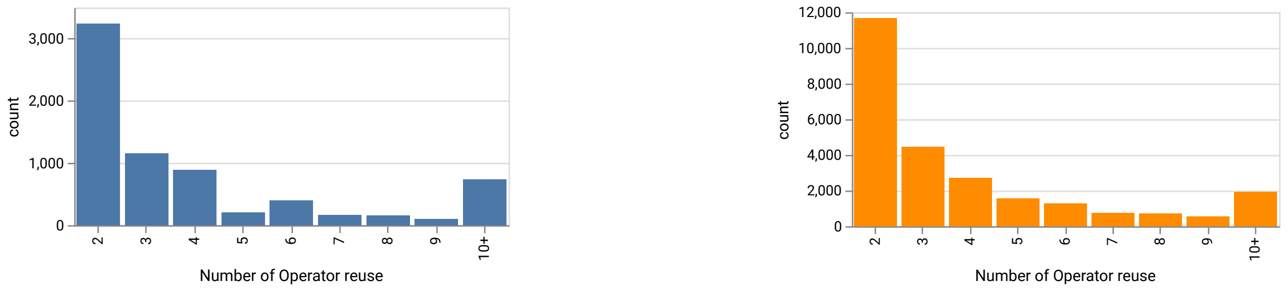


Figure 6: Distribution of number of operators that can benefit from reuse. (left) Data 100: $\mu = 5$, $\eta = 3$, $\sigma = 8$; (right) Github: $\mu = 7$, $\eta = 3$, $\sigma = 14$

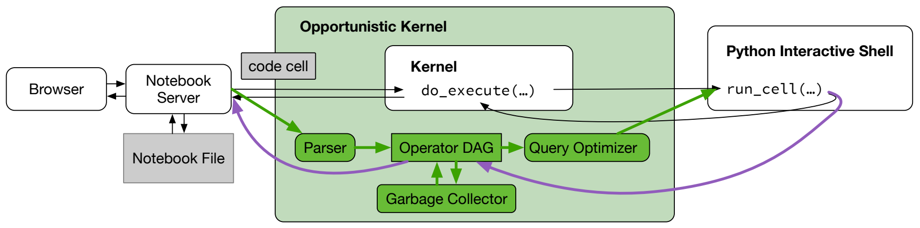


Figure 7: Opportunistic Evaluation Kernel Architecture.

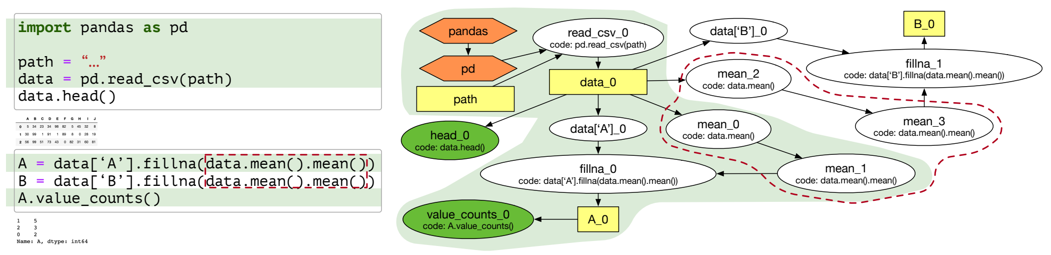
rendering of the notebook by Github, but further studies are needed to corroborate this hypothesis. Lastly, partial views are not nearly as prevalent as non-critical operators before an interaction, accounting only for $< 20\%$ of the interactions.

Reuse of Intermediate Results. Since dataframe queries are incrementally constructed, with subsequent queries building on top of previous ones, another common query optimization technique that is applicable is caching these intermediate results. To assess the opportunities to speed up queries by caching, we evaluate the number of times an operator is shared by different interactions but not stored as a variable by the user. Ideally, we would also have the execution times of the individual operators, which is not possible without a full replay. We present an initial analysis that only assesses the *existence* of reuse opportunities, as shown in Figure 6. Both the Data 100 and Github datasets have a median of 3 operators that can benefit from reuse.

Of the types of optimizations explored, operator reordering appears to be the most common. Thus, we focus our initial explorations of opportunistic evaluation on operator reordering for asynchronous execution during think time, while supporting preemption to interrupt asynchronous execution and prioritize interaction.

4 System Architecture

In this section, we introduce the system architecture for implementing our opportunistic evaluation framework for dataframe query optimization within Jupyter notebooks. At a high level, we create a custom Jupyter Kernel to intercept dataframe queries in order to defer, schedule, and optimize them transparently. The query execution engine uses an operator DAG representation for scheduling and optimizing queries and caching results, and is responsible for scheduling asynchronous query executions during think time. When new interactions arrive, the execution of non-critical operators is preempted and partial results are cached to resume execution during the next think time. A garbage collector periodically uncaches results corresponding to the DAG nodes to avoid memory bloat based on the likelihood of reuse.



```
import pandas as pd

path = "..."
data = pd.read_csv(path)
data.head()
```

	A	B	C	D	E	F	G	H	I	J
0	5	34	23	34	98	82	5	45	32	8
1	30	99	1	91	1	89	8	0	28	19
2	56	99	51	73	43	0	82	31	60	81

```
A = data['A'].fillna(data.mean().mean())
B = data['B'].fillna(data.mean().mean())
A.value_counts()
```

```
1    5
2    3
0    2
Name: A, dtype: int64
```

Figure 8: Example Code Snippet and Operator DAG.

4.1 Kernel Instrumentation

Figure 7 illustrates the round-trip communication between the Jupyter front-end and the Python interactive shell. The black arrows indicate how communication is routed normally in Jupyter, whereas the green and purple arrows indicate how we augment the Jupyter Kernel to enable opportunistic evaluation. First, when the code is passed from the front-end to the kernel, it is intercepted by the custom kernel that we created by wrapping the standard Jupyter kernel. As shown in the green box, the code is passed to a parser that generates a custom intermediate representation, the operator DAG. The operator DAG is then passed to the query optimizer to create a physical plan for the query to be executed. This plan in then passed to the Python interactive shell for execution. When the shell returns the result after execution, the result is intercepted by the custom kernel to augment the operator DAG with runtime statistics as well as partial results to be used by future queries, and the query results are passed back to the notebook server, as indicated by the purple arrows.

4.2 Intermediate Representation: Operator DAG

Figure 8 shows an example operator DAG constructed from the code snippet on the left. The orange hexagons are imports, yellow boxes are variables, ovals are operators, where green ovals are interactions. The operator DAG is automatically constructed by analyzing the abstract syntax tree of the code, in the parser component in Figure 7. We adopt the static single assignment form in our operator DAG node naming convention to avoid ambiguity of operator references, as the same operator can be invoked many times, either on the same or different dataframes. In the case that the operator DAG contains non-dataframe operators, we can simply project out the irrelevant operators by keeping only the nodes that are weakly connected to the `pandas` import node.

To see how the operator DAG can be used for optimization, consider two simple use cases:

Critical path identification. To identify the critical path to the interaction `A.value_counts()`, we can simply start at the corresponding node and traverse the DAG backwards to find all dependencies. Following this procedure, we would collect all nodes in the green region as the critical path to `A.value_counts()` (corresponding statements are highlighted in green on the left), slicing out the operators associated with the statement `B = data['B'].fillna(data.mean().mean())`, which does not need to be computed for the interaction.

Identifying repeated computation. Note that `data.mean().mean()` is a common subexpression in both `A` and `B`; recognizing this allows us to cache and reuse the result for `data.mean().mean()`, which is expensive since it requires visiting every element in the dataframe. We assume that operators are *idempotent*, i.e., calling the same operators on the same inputs would always produce the same results. Thus, descendants with identical code would contain the same results. Based on this assumption, we eliminate common subexpressions by starting at the root nodes and traversing the graph breadth first, merging any descendants with identical code. We then proceed to the descendants of the descendants and carry out the same procedure until the leaf nodes are reached. Following this procedure, we would merge `mean_0` with `mean_2` and `mean_1` with `mean_3` in the red dotted region in Figure 8.

We will discuss more optimizations in Section 5.

4.3 Operator Execution & Garbage Collector

When a notebook cell is executed, the opportunistic kernel first parses the code in the cell to add operators to the operator DAG described above. The DAG is then passed to the query optimizer, which will either immediately kick off the execution of interaction critical paths, if they are present in the DAG, or consider all the non-critical operators to determine what to execute next. We discuss optimizations for non-critical operators in Section 5.2.

After the last interaction is executed and the results are returned, the query optimizer will continue executing operators asynchronously until the entire DAG is executed. In the event that an interaction arrives while a non-critical operator is executing, we preempt the execution of the non-critical operator to avoid delaying the execution of the interaction critical path. We discuss optimizations for supporting effective preemption in Section 5.1.

While the kernel executes operators, a garbage collector (GC) is working in the background to uncache results in the operator DAG to control memory consumption. A GC event is triggered when memory consumption is above 80% of the maximum memory allocated to the kernel, at which point the GC inspects the operator DAG to uncache the set of operator results that are the least likely to speed up future queries. We discuss cache management in Section 5.2.

5 Optimization Framework

The opportunistic evaluation framework optimizes for interactive latency by deferring to think time the execution of operators that do not support interactions. The previous section describes how we use simple program analysis to identify the interaction critical path that must be executed to produce the results for an interaction. In this section, we discuss optimizations for minimizing the latency of a given interaction in Section 5.1 and optimizations for minimizing the latency of future interactions by leveraging think time in Section 5.2. We discuss how to model user behavior to anticipate future interactions in Section 5.3.

5.1 Optimizing Current Interactions

Given an interaction critical path, we can apply standard database optimizations for single queries to optimize interactive latency. For example, if the interaction operator is `head` (i.e., examining the first K rows), we can perform predicate pushdown to compute only part of the interaction critical path that leads to the top K rows in the final dataframe. The rest can be computed during think time in anticipation of future interactions.

The main challenge for optimizing interactive latency in opportunistic evaluation is the ability to effectively preempt the execution of non-critical operators. This preemption ensures that we avoid increasing the interactive latency due to irrelevant computation. The current implementation of various operators within pandas and other dataframe libraries often involves calling lower-level libraries that cannot be interrupted during their execution. In such cases, the only way to preempt non-critical operators is to abort their execution completely, potentially wasting a great deal of progress. We propose to overcome this challenge by partitioning the dataframe so that preemptions lead to, in the worst case scenario, only loss of the progress on the current partition.

Dataframe partitioning. Partitioning the dataframe in the opportunistic evaluation setting involves navigating the trade-off between the increase in future interactive latencies due to loss of progress during preemption and the reduction in operator latency due to missed holistic optimizations on the entire dataframe. In the setting where interactions are sparse, a single partition maximizes the benefit of holistic optimization while losing progress on the entire operator only occasionally due to preemption. On the other hand, if interactions are frequent and erratic, a large number of small partitions ensures progress checkpointing, at the expense of longer total execution time across all partitions. Thus, the optimal partitioning strategy is highly dependent on user behavior. We discuss how to model user behavior in Section 5.3.

Without a high-fidelity user interaction model, we can create unevenly sized partitions to handle the variability in the arrival rate of interactions. First, we create small partitions for the top and bottom K rows in the dataframe not only to handle the rapid succession of interactions but also to support partial-result queries involving `head` and `tail` that are prevalent in interactive dataframe workloads. Then, for the middle section of the dataframe, the partitions can reflect the distribution of think time such that the partition sizes are smaller at intervals where interactions are likely to be issued. For example, if the median think time is 20s and the operator's estimated execution time is 40s, it might be desirable to have smaller partitions after 50% of the rows have been processed.

The above strategy assumes sequential processing of every row in the dataframe. If, instead, the prevalent workload is working with a select subset of rows, then it is more effective to partition based on the value of the attributes that are commonly used for selection. Of course, partitioning is not necessary if computation started during think time does not block computation for supporting interactions.

Note that another important consideration in generating partial results is the selectivity of the underlying operators and whether they are blocking operators. For the former, we may need to employ a much larger partition simply to generate K results. For the latter, we may need to prioritize the generation of the aggregates corresponding to the groups in the top or bottom K (in the case of group-by), or to employ algorithms that prioritize the generation of the K first sorted results (in the case of sorting). In either case, the problem becomes a lot more challenging.

5.2 Optimizing Future Interactions Leveraging Think Time

Non-critical Operator scheduling. We now discuss scheduling non-critical operators. Recall that these operators are organized in a DAG built from queries. The job of our scheduler is to decide which *source* operators to execute. Source operators in the DAG are those whose precedent operators do not exist or are already executed. We assume an equal probability of users selecting any operator in the DAG to extend with an interaction.

The scheduler is optimized to reduce the interaction latency; we introduce the notion of an operator's *delivery cost* as the proxy for it. If an operator has not been executed yet, its delivery cost is the cost of executing the operator along with all of its unexecuted predecessors. Otherwise, the delivery cost is zero. Our scheduler prioritizes scheduling the source operator that can reduce the delivery cost across all operators the most. We define a utility function $U(s_i)$ to estimate the benefit of executing a source operator s_i. This function, for a node s_i is set to be the sum of the delivery cost for the source operator and all of its successors D_i:

$$U(s_i) = \sum_{j \in D_i} c_j \tag{3}$$

where c_j is the delivery cost for an operator j. Our scheduler chooses to execute the one with the highest $U(s_i)$. This metric prioritizes those operators that "influence" as many expensive downstream operators as possible.

Caching for reuse. When we are executing operators in the background, we store the result of each newly computed operator in memory. However, if the available memory (i.e., the memory budget) is not sufficient to store the new result, we need to recover enough memory by discarding materialized results of previously computed operators. If the discarded materialized results are needed by future operators, we will execute the corresponding operators to recompute them. Here, the optimization problem is to determine which materialized results should be discarded given the memory budget. Our system addresses this problem by systematically considering three aspects of a materialized result, denoted r_i: 1) the chance of r_i being reused, p_i, 2) the cost of recomputing the materialized result, k_i, and 3) the amount of memory it consumes, m_i. We estimate p_i by borrowing ideas from the LRU replacement algorithm. We maintain a counter T to indicate the last time any materialized result is reused and each materialized result is associated with a variable t_i that tracks the last time it is reused. If one materialized result r_i is reused, we increment the counter T by one and set t_i to T. We use the following formula to estimate p_i:

$$p_i = \frac{1}{T + 1 - t_i} \tag{4}$$

We see that the more recently a materialized result r_i is reused, the higher p_i is. We can use a cost model as in relational databases to estimate the recomputation cost k_i. We note that we do not always recompute an operator from scratch. Given that the other materialized results are in memory, our cost model estimates the recomputation cost by considering reusing existing materialized results. Therefore, we use the following utility function to decide which materialized result should be discarded.

$$O(r_i) = p_i \times \frac{m_i}{k_i} \tag{5}$$

Here, $\frac{m_i}{k_i}$ represents the amount of memory we can spare per unit of recomputation cost to pay. The lower $\frac{m_i}{k_i}$ is, the more likely we discard r_i. Finally, our algorithm will discard the r_i with the lowest $O(r_i)$ value.

Speculative materialization. Our system not only considers caching results generated by users' programs, but also speculatively materializes and caches results that go beyond what users specify, to be used by future operators. One scenario we observed is that users intend to explore the data by changing the value of a filter repeatedly. In this case, we can materialize the intermediate output results before we apply the filter and when users modify the filter, we can reuse the saved results without computing them from scratch. The downside of this approach is that it can increase the latency of computing an interaction when the think time is limited. Therefore, we enable this optimization only when users' predicted think time of writing a new operator is larger than the time of materializing the intermediate states.

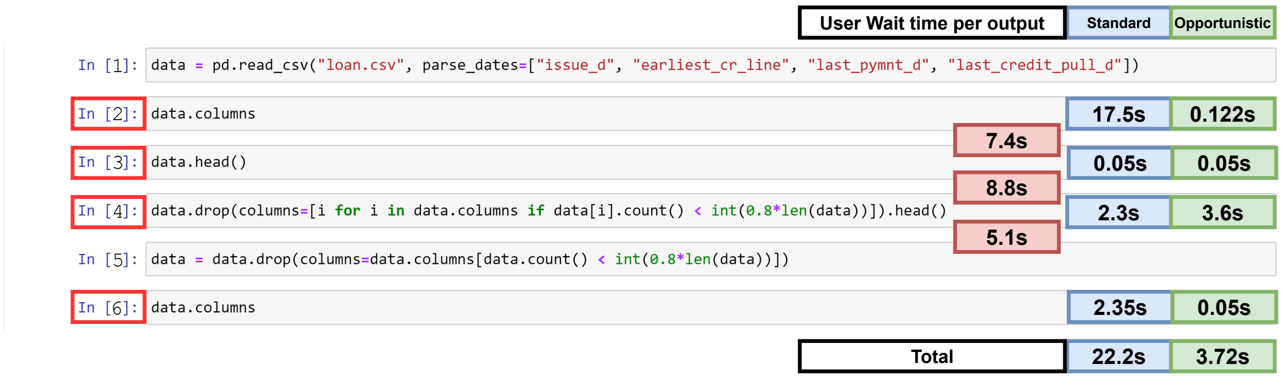
5.3 Prediction of User Behavior

The accurate prediction of user behavior can greatly improve the efficacy of opportunistic evaluation. Specifically, we need to predict two types of user behavior: think time and future interactions. Section 3.1 described some preliminary statistics that can be used to construct a prior distribution for think time. As the system observes the user work, this distribution can be updated to better capture the behavior of the specific user, as we expect the distribution of think time to vary greatly based on the dataset, task, user expertise, and other idiosyncrasies. These workload characteristics can be factored into the think time model for more accurate prediction. This think time model can be used by the optimizer to decide the size of dataframe partitions to minimize progress loss due to preemption or to schedule non-critical operators whose expected execution times are compatible with the think time duration.

To predict future interactions, we can use the models from Yan et al. [17]. These models are trained on a large corpus of data science notebooks from Github. Since future interactions often build on existing operators, we can use the future interaction prediction model to estimate the probabilities of non-critical operators in the DAG leading to future interactions, which can be used by the scheduler to pick non-critical operators to execute next. Let p_j be the probability of the children of an operator j being an interaction. We can incorporate p_j into the utility function in Equation 3 to obtain the updated utility function:

$$U_p(s_i) = \sum_{j \in D_i} c_j \times p_j \tag{6}$$

Of course, the benefits of opportunistic evaluation can lead to modifications in user behavior. For example, without opportunistic evaluation, a conscientious user might self-optimize by avoiding specifying expensive non-critical operators before interactions, potentially at the cost of code readability. When self-optimization is no longer necessary when authoring queries, the user may choose to group similar operators for better code readability and maintenance, thus creating more opportunities for opportunistic evaluation optimizations.



```
In [1]: data = pd.read_csv("loan.csv", parse_dates=["issue_d", "earliest_cr_line", "last_pymnt_d", "last_credit_pull_d"])

In [2]: data.columns

In [3]: data.head()

In [4]: data.drop(columns=[i for i in data.columns if data[i].count() < int(0.8*len(data))]).head()

In [5]: data = data.drop(columns=data.columns[data.count() < int(0.8*len(data))])

In [6]: data.columns
```

Figure 9: An example notebook. Cells that show an output are indicated with a red box.

6 Case Study

In this section, we evaluate how opportunistic evaluation will impact the end user through a case study. Figure 9 shows an excerpt from the original notebook, taken from a Kaggle competition (https://www.kaggle.com/c/home-credit-default-risk).

In this case study, the data scientist first read in the file, and was forced to immediately wait. Then, the user wanted to see the columns that exist in the dataset. This is often done when the data scientist first encounters a dataset. They therefore printed the first 5 lines with `data.head()`. This inspection is important for data validation: the data scientist wanted to ensure that the data was parsed correctly during data ingestion. After these two data validation steps, the data scientist noticed that there were a significant number of `null` values in multiple columns.

The cell labeled `In[4]` shows how the data scientist solved the `null` values problem: they decided to drop any column that does not have at least 80% of its values present. Notice that the data scientist first wanted to see what the results of the query would look like before they executed it, so they added a `.head()` to the end of the query that drops the columns. Likely this was done during debugging, where many different, but similar queries were attempted until the desired output was achieved. The query was then repeated to overwrite the `data` variable. An important note here is that the full dataset is lost at this point due to the overwriting of the `data` variable. The data scientist will need to reread the file if they want access to the full dataset again. After dropping columns with less than 80% of their values present, the data scientist double-checked their work by inspecting the `columns` of the overwritten `data` dataframe. Next, we evaluate the benefits of the opportunistic evaluation approach by determining the amount of synchronous wait time saved by leveraging think time.

To evaluate opportunistic evaluation in our case study, think time was injected into the notebook from the distribution presented in Figure 3. We found that the time that the hypothetical data scientist spent waiting on computation was almost none: the `read_csv` phase took 18.5 seconds originally, but since the output of the `columns` and `head` were prioritized, they were displayed almost immediately (122ms). The data scientist then looked at the two outputs from `columns` and `head` for a combined 16.2 seconds. This means the data scientist synchronously waited on the `read_csv` for approximately 1.3 seconds. Next, the user had to wait another 2.3 seconds for the columns with less than 80% of their values present to be dropped. Without opportunistic evaluation, the user would have to pay this time twice, once to see the first 5 lines with `head` and again to see the `data.columns` output in cell `In[6]`.

7 Related Work

Recently, researchers and developers have begun to turn their attention to the optimization, usability, and scalability of dataframes as the community begins to recognize its important role in data exploration and analysis. Some of these issues are brought on by the increasingly complex API and ever-growing data sizes. Pandas itself has best practices for performance optimization embedded within its user guide [4]. However, enabling these optimizations often requires a change to the user's behavior.

Many open source systems attempt to provide improved performance or scalability for dataframes. Often, this means only supporting dataframe functionalities that are simple to parallelize (e.g., Dask [3]), or supporting only those operations which can be represented in SQL (e.g. Koalas [5] and SparkSQL [2]). Our project, Modin [6], is the only open source system with an architecture that can support all dataframe operators.

In the research community, there are multiple notable papers that have tackled dataframe optimization through vastly different approaches. Sinthong et al. propose AFrame, a dataframe system implemented on top of AsterixDB by translating dataframe APIs into SQL++ queries that are supported by AsterixDB [15]. Another work by Yan et al. aims to accelerate EDA with dataframes by "auto-suggesting" data exploration operations [17]. Their approach has achieved considerable success in predicting the operations that were actually carried out by users given an observed sequence of operations. More recently, Hagedorn et. al. designed a system for translating pandas operations to SQL and executing on existing RDBMSs [9]. In a similar vein, Jindal et. al. built a system called Magpie for determining the optimal RDBMS to execute a given query [11]. Finally, Sioulas et. al. describe techniques for combining the techniques from recommendation systems to speculatively execute dataframe queries [16].

Our proposed approach draws upon a number of well established techniques from the systems, PL, and DB communities. Specifically, determining and manipulating the DAG of operators blends control flow and data flow analysis techniques from the PL community [8]. The optimization of dataframe operators draws inspiration from battle-tested database approaches such as predicate pushdown, operator reordering, multi-query optimization, and materialized views [10], as well as compiler optimizations such as program slicing and common subexpression elimination. Furthermore, we borrow from the systems literature on task scheduling to take enable asynchronous execution of dataframe operators during think time.

8 Conclusion & Future Work

We proposed opportunistic evaluation, a framework for accelerating interactions with dataframes. Interactive latency is critical for iterative, human-in-the-loop dataframe workloads for supporting data validation, both for ML and for EDA. Opportunistic evaluation significantly reduces interactive latency by 1) prioritizing computation directly relevant to the interactions and 2) leveraging think time for asynchronous background computation for non-critical operators that might be relevant to future interactions. We have shown, through empirical analysis, that current user behavior presents ample opportunities for optimization, and the solutions we propose effectively harness such opportunities.

While opportunistic evaluation addresses data validation prior to model training, data validation challenges are present in other parts of the end-to-end ML workflow. For example, after a trained model has been deployed, it is crucial to monitor and validate online data against the training data in order to detect data drift, both in terms of distribution shift and schema changes. A common practice to address data drift is to retrain the model on newly observed data, thus introducing data drift into the data pre-processing stage of the end-to-end ML workflow. Being able to adapt the data validation steps in a continuous deployment setting to unexpected data changes is an open challenge.

References

[1] Google Colab. `colab.research.google.com`.

[2] PySpark 2.4.4 Documentation: pyspark.sql module. `http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions`.

[3] Dask Documentation. `https://docs.dask.org/en/latest/`, 2020.

[4] Enhancing performance, Pandas Documentation. `https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html`, 2020.

[5] Koalas: pandas api on apache spark. `https://koalas.readthedocs.io/en/latest/`, 2020.

[6] Modin Documentation. `https://modin.readthedocs.io/en/latest/`, 2020.

[7] Pandas API reference. `https://pandas.pydata.org/pandas-docs/stable/reference/index.html`, 2020.

[8] K. Cooper and L. Torczon. *Engineering a compiler*. Elsevier, 2011.

[9] S. Hagedorn, S. Kläbe, and K.-U. Sattler. Putting pandas in a box. *The Conference on Innovative Data Systems Research (CIDR)*.

[10] J. M. Hellerstein, M. Stonebraker, J. Hamilton, et al. Architecture of a database system. *Foundations and Trends® in Databases*, 1(2):141–259, 2007.

[11] A. Jindal, K. V. Emani, M. Daum, O. Poppe, B. Haynes, A. Pavlenko, A. Gupta, K. Ramachandra, C. Curino, A. Mueller, et al. Magpie: Python at speed and scale using cloud backends. *The Conference on Innovative Data Systems Research (CIDR)*.

[12] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[13] S. Macke, H. Gong, D. J.-L. Lee, A. Head, D. Xin, and A. Parameswaran. Fine-grained lineage for safer notebook interactions. *Proceedings of the VLDB Endowment*, 2021.

[14] D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. Parameswaran. Towards scalable dataframe systems. *Proceedings of the VLDB Endowment*, 13(11).

[15] P. Sinthong and M. J. Carey. Aframe: Extending dataframes for large-scale modern data analysis. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 359–371. IEEE, 2019.

[16] P. Sioulas, V. Sanca, I. Mytilinis, and A. Ailamaki. Accelerating complex analytics using speculation. *The Conference on Innovative Data Systems Research (CIDR)*.

[17] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1539–1554, 2020.

Responsible AI Challenges in End-to-end Machine Learning

Steven Euijong Whang, Ki Hyun Tae, Yuji Roh, Geon Heo
KAIST
{swhang, kihyun.tae, yuji.roh, geon.heo}@kaist.ac.kr

Abstract

Responsible AI is becoming critical as AI is widely used in our everyday lives. Many companies that deploy AI publicly state that when training a model, we not only need to improve its accuracy, but also need to guarantee that the model does not discriminate against users (fairness), is resilient to noisy or poisoned data (robustness), is explainable, and more. In addition, these objectives are not only relevant to model training, but to all steps of end-to-end machine learning, which include data collection, data cleaning and validation, model training, model evaluation, and model management and serving. Finally, responsible AI is conceptually challenging, and supporting all the objectives must be as easy as possible. We thus propose three key research directions towards this vision – depth, breadth, *and* usability *– to measure progress and introduce our ongoing research. First, responsible AI must be deeply supported where multiple objectives like fairness and robust must be handled together. To this end, we propose FR-Train, a holistic framework for fair and robust model training in the presence of data bias and poisoning. Second, responsible AI must be broadly supported, preferably in all steps of machine learning. Currently we focus on the data pre-processing steps and propose Slice Tuner, a selective data acquisition framework for training fair and accurate models, and MLClean, a data cleaning framework that also improves fairness and robustness. Finally, responsible AI must be usable where the techniques must be easy to deploy and actionable. We propose FairBatch, a batch selection approach for fairness that is effective and simple to use, and Slice Finder, a model evaluation tool that automatically finds problematic slices. We believe we scratched the surface of responsible AI for end-to-end machine learning and suggest research challenges moving forward.*

1 Introduction

Responsible AI is becoming critical as machine learning becomes widespread in our everyday lives. Companies including Google [2], Microsoft [3], and IBM [5] publicly state that AI not only needs to be accurate, but also used and developed, evaluated, and monitored for trust. Although there is no universally agreed notion for responsible AI, the major objectives include fairness, robustness, explainability, transparency, and accountability.

The usual starting point is to support responsible AI only in model training, but this is not sufficient. For example, if the training data is biased towards a specific population, there is a fundamental limit into how much the trained model can avoid being biased as well even using the best fair training algorithms. Instead, we may need to address the root cause starting from data collection where we need to construct an unbiased dataset.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

We would thus like to support responsible AI in all steps of end-to-end machine learning [8, 37]. Before model training, the key steps are data collection, data cleaning, and validation. After model training, there are model evaluation, and model management and serving. In addition, since supporting all the responsible AI objectives is already conceptually challenging, it is important to make these techniques easy to use as well.

To this end, we propose three research directions – *depth*, *breadth*, and *usability* – and present our contributions. First, we need to deeply support responsible AI where multiple objectives are addressed together. We present FR-Train [28], the first holistic framework for fair and robust model training. Second, we need to broadly support responsible AI in all machine learning steps. We present two systems that focus on data pre-processing: Slice Tuner [34] is a selective data acquisition framework for fair and accurate models, and MLClean [33] is a data cleaning framework that also improves fairness and robustness. Third, we need responsible AI to be usable and actionable. We present two systems: FairBatch [29] is an easy-to-deploy batch selection technique for model training that improves fairness, and Slice Finder [13, 14] automatically evaluates a model by finding problematic slices where it underperforms. Our work only scratches the surface of responsible AI for end-to-end machine learning, and we believe that setting the three research directions is useful to measure progress.

We introduce the responsible AI research landscape in Section 2. We then discuss our systems for depth, breadth, and usability in Sections 3, 4, and 5, respectively. Finally, we suggest open challenges in Section 6.

2 Responsible AI Research Landscape

We provide a brief history of responsible AI and discuss the research landscape. Responsible AI is also known as Trustworthy AI and has recently been promoted by Google [2], Microsoft [3], and IBM [5] among others as a critical issue when using AI in practice. The key objectives include fairness, robustness, explainability, transparency, and accountability. Among the objectives, we focus on fairness and robustness because they are both closely related to the training data. The other objectives are also important, but currently outside our scope.

Fairness is the problem of not discriminating against users and has gained explosive interest in the past decade [7, 35]. An article that popularized fairness was the 2016 ProPublica report [6] on the COMPAS software, which is used in US courts to predict a defendant's recidivism (reoffending) rate. COMPAS is convenient, but is known to overestimate black people's recidivism risk compared to white people. Recently, various unfairness mitigation techniques [9] have been proposed and can be categorized as pre-processing, in-processing, or post-processing depending on whether the techniques are applied before, during, or after model training, respectively.

Robustness is the problem of preventing or coping with adversarial attacks. In particular, model training against data poisoning has been heavily studied in the past decade [15, 31]. Nowadays datasets are easier to publish using tools like Kaggle and Google Dataset Search [11], which means that it is easier to disseminate poisoned data as well. The data can then be harvested by Web crawlers of unsuspecting victims and used for model training. While the basic poisoning attacks involve simple labeling flipping (e.g., change a positive label to be negative), recent poisoning attacks are becoming increasingly sophisticated. The possible defenses include sanitizing the data before model training or making the model training accurate despite the poisoning.

In practice, machine learning is not just about model training, but involves multiple steps as demonstrated by end-to-end systems like TensorFlow Extended (TFX) [8] and MLFlow [37]: data collection, data cleaning and validation, model training, model evaluation, and model management and serving. Hence, responsible AI is not just a model training issue, but relevant to all of the above steps. The data management community has recently been addressing the data aspect of responsible AI in end-to-end machine learning [25, 23, 26, 12, 27, 32, 36].

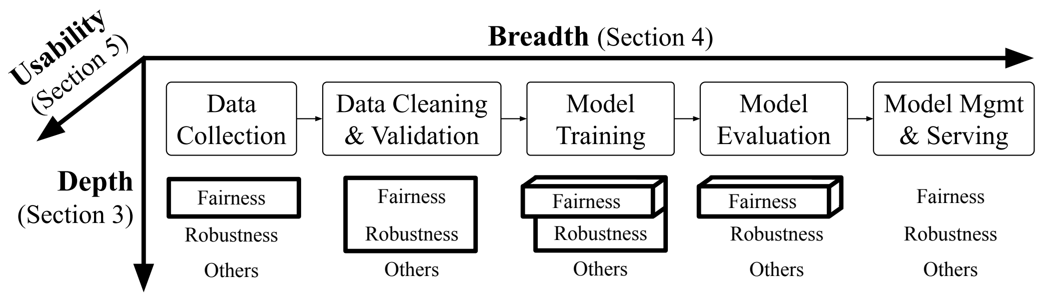


Figure 1: The three research directions – depth, breadth, and usability – for fully supporting the responsible AI objectives (fairness, robustness, and others) in addition to accuracy in end-to-end machine learning. The highlighted parts show our contributions: Slice Tuner [34] addresses fairness in data collection; MLClean [33] addresses fairness and robustness in data cleaning; FR-Train [28] addresses fairness and robustness in model training; FairBatch [29] addresses usability for fairness in model training; and Slice Finder [13, 14] addresses usability for fairness in model evaluation.

The current research landscape naturally leads to the three key research directions we propose – *depth*, *breadth*, and *usability* – as shown in Figure 1. First, it is important to support many responsible AI objectives at each step. Second, we need to broadly support responsible AI in as many steps as possible, from data collection to model serving. Third, we need these techniques to be usable and actionable by machine learning users. We highlight the responsible AI objectives in Figure 1 where we propose solutions.

3 Deep Responsible AI

We discuss deeply supporting responsible AI, which means that we would like to address multiple objectives together. We re-emphasize that each objective is currently being heavily studied. For model fairness, there is an extensive literature in the machine learning and fairness communities on mitigating unfairness before, during, or after model training [7, 35, 9]. For model robustness, both the machine learning and security communities are proposing various data sanitization and robust training techniques [22, 31]. However, we believe that responsible AI requires both fairness and robustness instead of just one. In addition, addressing one objective at a time is not ideal as we discuss later. Fairness and robustness are also closely related because their problems originate from the training data: biased data causes unfairness while poisoned data decreases model accuracy. This motivation leads us to propose FR-Train [28], the first holistic framework for fair and robust training.

Fairness is a subjective notion, and many definitions have been proposed [35] where they can be categorized depending on what information is used: the classifier, the sensitive attribute (e.g., race or gender), and training labels. For example, individual fairness only uses the classifier and means that similar individuals must have similar predictions. Demographic parity [17] (or disparate impact) uses the classifier and the protected attribute and means that different sensitive groups (e.g., black and white populations) have similar positive prediction rates. That is, $P(\hat{Y}=1|Z=0) \approx P(\hat{Y}=1|Z=1)$ where $\hat{Y}$ is a prediction and Z is a binary sensitive attribute. Equalized odds [18] uses all three pieces of information and is similar to demographic parity, except that the probabilities are conditioned on the label. That is, $P(\hat{Y}=1|Z=0,Y=l) \approx P(\hat{Y}=1|Z=1,Y=l)$ where Y is the label. In this section, we use demographic parity and measure it using the formula $DP := \min\left(\frac{P(\hat{Y}=1|Z=0)}{P(\hat{Y}=1|Z=1)}, \frac{P(\hat{Y}=1|Z=1)}{P(\hat{Y}=1|Z=0)}\right)$ where a higher value close to 1 means better fairness.

We now explain why addressing fairness and robustness together is important using a concrete example. In Figure 2, suppose there are two sensitive groups black and white, and that there are ten people of two races: white (denoted as 'w') and black (denoted as 'b'). Let us assume the boxes indicates positive labels and that we want to train a threshold classifier that divides the individuals using a single feature X where those on the left have

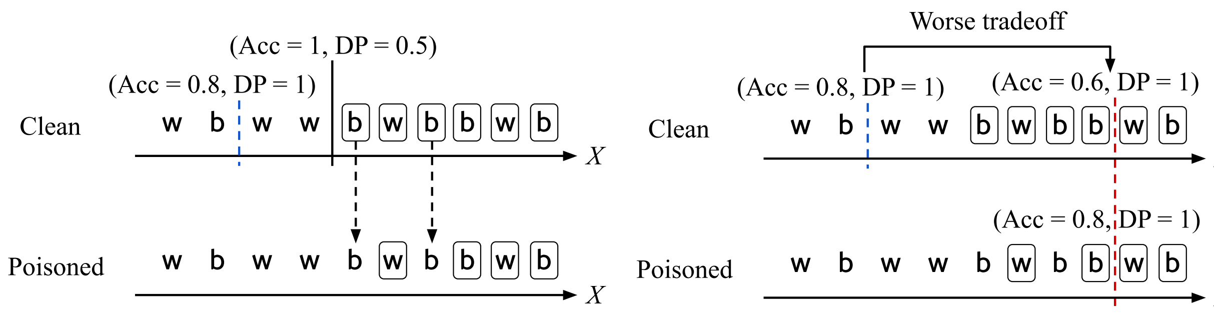


Figure 2: (Left) Accurate (black solid line) and fair (blue dotted line) classifiers on clean data followed by data poisoning. (Right) A fair classifier trained on poisoned data (red dotted line) is evaluated on clean data, showing a worse accuracy-fairness tradeoff than the fair classifier trained on clean data.

negative predictions (e.g., do not reoffend) while those on the right have positive predictions. On clean data, a vanilla classifier can obtain perfect accuracy by dividing between the fourth and fifth individuals (Figure 2 solid line classifier). However, the demographic parity DP is not perfect where $P(\hat{Y}=1|Z=w)=\frac{2}{5}=0.4$, $P(\hat{Y}=1|Z=b)=\frac{4}{5}=0.8$, and $DP := \min\left(\frac{0.4}{0.8},\frac{0.8}{0.4}\right)=0.5$. Suppose a fair classifier maximizes accuracy with perfect DP. One can find such a classifier by dividing between the second and third individuals (Figure 2 blue dotted line classifier). While $DP=1$, the accuracy is 0.8 because two white people are now misclassified.

Now suppose we poison the clean data by using the standard method of flipping labels [24]. On the bottom of the left side of Figure 2, the fifth and seventh individuals are now incorrectly labeled as negative. There are three ways to handle the poisoned data: (1) do nothing and perform fair training only as usual, (2) take a two-step approach and perform data sanitization followed by fair training using existing methods, and (3) take a holistic approach for fair and robust training. Let us first see what happens if we take the first approach. We can train a fair classifier on the poisoned data with perfect DP by dividing between the eighth and ninth individuals (bottom of the right side of Figure 2, red dotted line classifier). In that case, we will have perfect DP, but an accuracy of 0.8 on poisoned data. However, if this classifier is deployed in the real world, it will effectively be used on clean data. This scenario is plausible for any application that serves real customers. However, simply using the same classifier on clean data results in a worse tradeoff of fairness and accuracy where DP remains the same, but the accuracy reduces to 0.6. Hence, ignoring poisoning may lead to strictly worse accuracy and fairness results. In reference [28], we also empirically show that the two-step solution is ineffective. The intuition is that an existing fairness-only or robustness-only technique cannot easily distinguish data poisoning from bias in the data and ends up removing all or none of the problematic data.

We thus propose FR-Train to take a holistic approach for fair and robust training. Figure 3 shows the architecture of FR-Train. On the top, there is a classifier (e.g., predicts recidivism) that competes with a discriminator for fairness that predicts the sensitive attribute (e.g., the race) based on the predictions. This adversarial training is similar to Adversarial Debiasing [38], a state-of-the-art fairness-only training algorithm. The below part is the novel addition where there is a discriminator for robustness that distinguishes the possibly-poisoned training set with a validation set that is known to be clean. The clean validation set is small and can be constructed using crowdsourcing and conventional quality control techniques including majority voting. Hence, the classifier needs to be both fair and robust to compete with the two discriminators. Finally, the predictions of the robustness discriminator are used to reweight training set examples where cleaner examples get higher weights. Initially, these weights are not useful because the robustness discriminator is not accurate. However, as the training progresses, the discriminator becomes accurate, and the weights are used by the classifier.

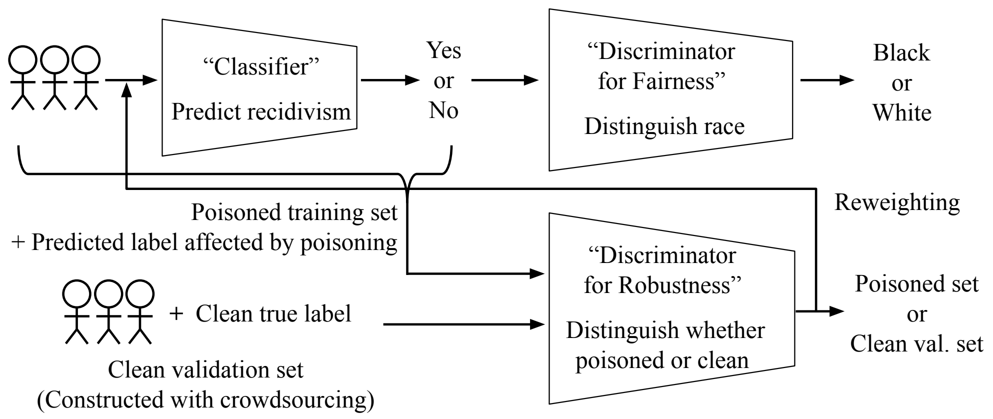

"Classifier"
Predict recidivism
Yes or No
"Discriminator for Fairness"
Distinguish race
Black or White
Poisoned training set
+ Predicted label affected by poisoning
Reweighting
+ Clean true label
Clean validation set
(Constructed with crowdsourcing)
"Discriminator for Robustness"
Distinguish whether poisoned or clean
Poisoned set or Clean val. set

Figure 3: The FR-Train architecture and how it can be used for recidivism prediction.

In reference [28], we present a mutual information-based interpretation of FR-Train's architecture. To give an intuition, perfect fairness means that the mutual information between the model's prediction and the sensitive attribute is 0. Similarly, satisfying robustness can be expressed using mutual information. In this case, perfect robustness means that the poisoned data distribution is indistinguishable from the clean data distribution (i.e., validation set). FR-Train minimizes both of the mutual information values and the classifier loss. We perform experiments on synthetic and real datasets and train a classifier on poisoned data and evaluate it on clean data. As a result, FR-Train is the only approach that achieves both high accuracy and fairness while the other baselines either have poor fairness or accuracy.

4 Broad Responsible AI

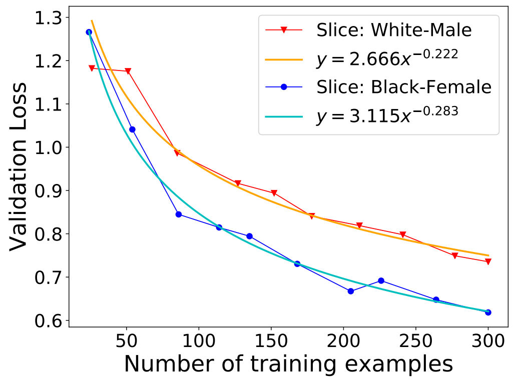
In addition to supporting responsible AI in model training, we would also like to broadly support it across many steps in end-to-end machine learning. While most of the fairness and robustness literature focus on model training, there needs to be more focus on other machine learning steps as well. Recently, FairPrep [30] was proposed to support fairness in all steps of data pre-processing before model training. Also for an extensive coverage of data collection and quality techniques for machine learning, please refer to a survey [27] and tutorial [36]. Here we also focus on data pre-processing and present two contributions: Slice Tuner [34] is a selective data acquisition framework for maximizing fairness and accuracy, and MLClean [33] is a data cleaning framework for addressing both fairness and robustness in addition to accuracy.

4.1 Selective Data Acquisition for Fair and Accurate Models

As machine learning is used in various applications, one of the critical bottlenecks is acquiring enough data so that the trained model is both accurate and fair. Nowadays, there are many ways to acquire data including dataset discovery, crowdsourcing, and simulator-based data generation. Data acquisition is not the same as active learning, which labels existing data. Instead, our focus is on acquiring new data along with its labels.

However, blindly acquiring data is not the right approach. Let us first divide the data into subsets called slices. Suppose that the slices are customer purchases by various regions: America, Europe, APAC, and so on. Among them, if we already have enough America data, acquiring more America data is not only unhelpful, but may also bias the data and have a negative effect on the model accuracy on the other slices.

Instead, we want to acquire possibly different amounts of data per slice in order to maximize accuracy and fairness. To measure accuracy, we use loss functions like logistic loss. For fairness, we use equalized error rates [35], which states that the losses of slices must be similar. This notion of fairness is important to any



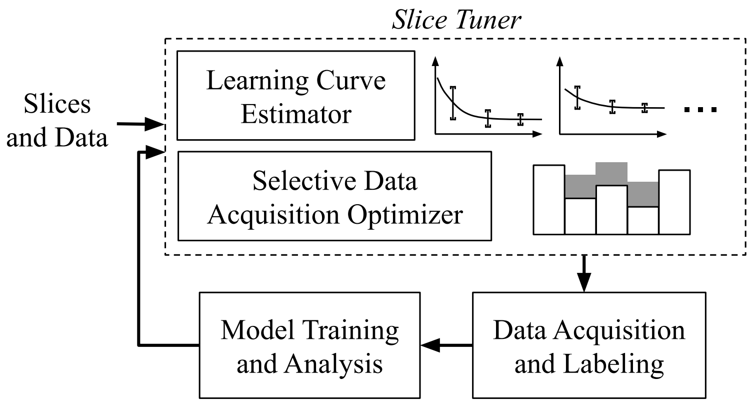

Slice Tuner
Slices and Data
Learning Curve Estimator
Selective Data Acquisition Optimizer
Model Training and Analysis
Data Acquisition and Labeling

Figure 4: (a) Learning curves on two slices of the UTKFace dataset [39]. (b) Slice Tuner architecture.

application that should not discriminate its customers by service quality. A waterfilling approach is a good start where we simply acquire data so that the slices have similar sizes. However, this approach is not optimal because some slices may need more data to obtain the same model loss as other slices.

Our key approach is to generate for each slice a *learning curve*, which estimates the model loss on that slice given more labeled data. Multiple studies [19, 16] show that a learning curve is best fit using a power-law function. Figure 4 (a) shows two actual learning curves generated on two race-gender slices of a real dataset called UTKFace [39]. We can use these learning curves to estimate how much data must be acquired per slice.

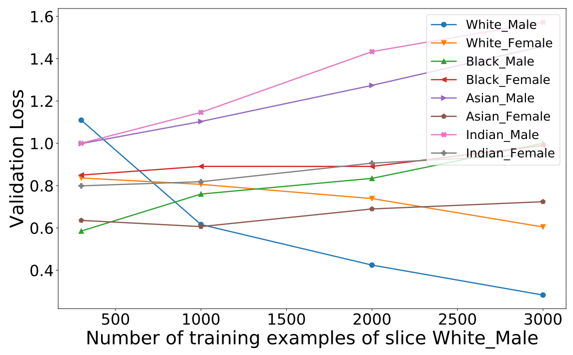
Assuming that the learning curves are perfectly reliable (we discuss how to deal with unreliable curves later), we can determine the amounts of data to acquire to minimize the total loss and unfairness of slices by solving the following convex optimization problem:

$$\min \sum_{i=1}^{n} b_i(|s_i| + d_i)^{-a_i} + \lambda \sum_{i=1}^{n} \max\left\{0, \frac{b_i(|s_i| + d_i)^{-a_i}}{A} - 1\right\} \text{ subject to } \sum_{i=1}^{n} C(s_i) \times d_i = B$$

where $\{s_i\}_{i=1}^n$ are the slices, $\{d_i\}_{i=1}^n$ are the amounts of data to acquire, A is the average loss of slices, $C(s_i)$ is the cost function for acquiring an example for s_i, and B is a cost budget. The first term in the objective function minimizes the total loss while the second term minimizes the unfairness by penalizing slices that have higher-than-average losses. The two terms are balanced using λ. By acquiring more data for slices with higher losses, we eventually satisfy equalized error rates. Slice Tuner's architecture is shown in Figure 4 (b) where we perform selective data acquisition on input slices. The runtime bottleneck is the time to actually acquire data.

We now address the key challenge of handling unreliable learning curves. Learning curves are not perfect because slices may be too small for accurate estimations. Even worse, acquiring data for one slice may "influence" others. Figure 5 (a) shows how acquiring data for the slice White-Male increases or even decreases the model's loss on other slices for UTKFace. The intuition is that the acquired data of one slice pushes the decision boundary of the model, which in turn changes the losses of other slices (Figure 5 (b)).

The solution is to iteratively update the learning curves. But how often should we iterate? On one hand, each iteration is expensive and involves multiple model trainings and curve fittings, even though we use amortization techniques [34]. On the other hand, we do not want to use inaccurate learning curves. Our algorithm works as follows. We first ensure a minimum slice size to draw some learning curve. In practice, having tens of examples is enough for this step. Next, we repeat two steps until we run out of budget: (1) acquire data as long as the estimated influence is not large enough and (2) re-fit the learning curves. The remaining problem is estimating influence. We propose a proxy called imbalance ratio change where imbalance ratio represents bias and is the ratio between the largest and smallest slice sizes. The intuition is that a change in imbalance ratio among slices



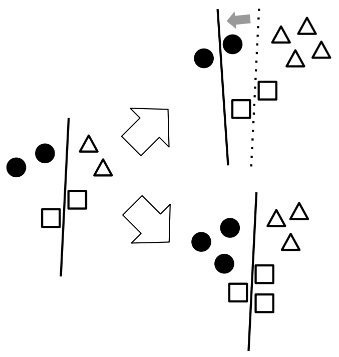


Figure 5: (a) Data acquisition on the slice White-Male influencing the losses on the other slices for the UTKFace dataset. (b) To give an intuition, say there are three slices where shape indicates slice, and color indicates label. (Top) If we only increase the triangles, the decision boundary may shift to the left due to the new bias, changing the losses of the other slices. (Bottom) If we evenly increase the data for all slices, the bias does not change, and there is little influence among the slices.

causes influence. In Figure 5 (b) adding two triangles results in a shifted decision boundary where the imbalance ratio increases from $\frac{2}{2} = 1$ to $\frac{4}{2} = 2$. On the other hand, if we evenly increase the slices, the decision boundary does not shift, and the imbalance ratio does not change much either.

In reference [34], we provide more details on the algorithms and also perform experiments on real datasets. We show that Slice Tuner has lower loss and unfairness compared to two baselines: uniformly acquiring the same amounts of data per slice and waterfilling. We also make the same comparison when the slices are small and only have tens of examples. Here the learning curves are very noisy and thus unreliable. Interestingly, Slice Tuner still outperforms the baselines because it can still leverage the relative loss differences among the learning curves. As more data is acquired, Slice Tuner performs even better with more reliable learning curves. In the worst case when the learning curves are completely random, we expect Slice Tuner to perform similarly to one of the baselines.

4.2 Data Cleaning for Accurate, Fair, and Robust Models

Another important place to support responsible AI is data cleaning [20] where the input data needs to be validated and fixed before it is used for model training. Historically, multiple communities – data management, machine learning (model fairness), and security – have been investigating this problem under the names of data cleaning, unfairness mitigation, and data sanitization, respectively. Unfortunately, not much is known how the different techniques can be used together when a dataset is dirty, biased, and poisoned at the same time.

MLClean is a unified cleaning framework that performs data cleaning, data sanitization, and unfairness mitigation together. A key insight is that these three operations have dependencies and must be executed in a certain order for the best performance. As shown in MLClean's architecture in Figure 7, data sanitization and cleaning are performed together followed by unfairness mitigation. Data sanitization can be considered a stronger version of cleaning because it defends against adversarial poisoning instead of just noise. In addition, data cleaning and sanitization may affect the bias of data while unfairness mitigation that performs example reweighting does not affect the correctness of cleaning and sanitization.

As a running example, suppose we run MLClean on our examples with equal weights of 1. Say that data sanitization clusters examples and removes anomalies while data cleaning performs entity resolution. The two operations can be naturally combined by generating clusters and running entity resolution within each cluster, assuming that examples across clusters do not match. Clustering examples before resolution is a common

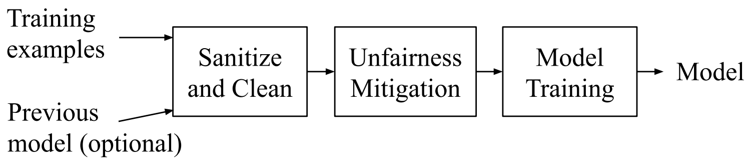


Figure 6: The MLClean architecture where data sanitization and data cleaning are performed together followed by unfairness mitigation. Optionally, a previous model can be used by any of the three operations.

ID	Weight	Name	Gender	Age	Label
e_1	1.0	John	M	20	1
e_2	1.0	Joe	M	20	0
e_3	1.0	Joseph	M	20	0
e_4	1.0	Sally	F	30	1
e_5	1.0	Sally	F	40	0
e_6	1.0	Sally	F	300	1

Table 4: Six examples where e_2 and e_3 are duplicates (dirty), and e_6 has an anomalous age (poisoned).

operation in entity resolution for narrowing down matching candidates. Figure 7 shows how the initial six examples are clustered into $\{e_1, e_2, e_3\}$ and $\{e_4, e_5\}$ (e_6 is considered an outlier), and then e_2 and e_3 are merged together into e_{23} with a summed weight of 2. For unfairness mitigation, suppose we reweight [21] the examples such that demographic parity (defined in Section 3) is satisfied for the sensitive groups men and women. We can make the (weighted) positive prediction rates the same by adjusting e_{23}'s weight from 2 to 1. As a result, the (weighted) positive prediction rates for men and women have the same value of $\frac{1.0}{1.0+1.0} = 0.5$.

In reference [33], we compare MLClean with other baselines that use a strict subset of the operations data sanitization, data cleaning, and unfairness mitigation or use all three operations, but in a different order than MLClean. On real datasets, MLClean has the best model accuracy and fairness, demonstrating that all three operations are necessary for the best results. In addition, MLClean is faster than baselines that use the three operations in different orders, which means that utilizing the dependencies among the operations is important.

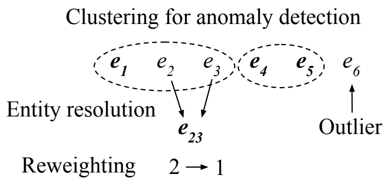
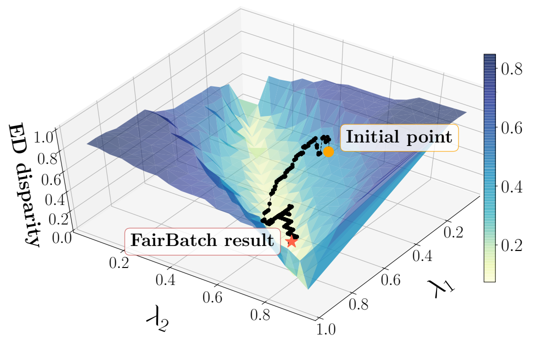

Clustering for anomaly detection
e_1 e_2 e_3 e_4 e_5 e_6
Entity resolution
e_{23}
Outlier
Reweighting
2 → 1

Figure 7: MLClean running on our examples.

5 Usable Responsible AI

The final pillar of responsible AI is making it usable and actionable to all machine learning users. While usability is not always the main focus in machine learning, it is especially relevant for responsible AI because the various objectives are already conceptually challenging to understand, so the deployment must be made as easy as possible. We thus propose two systems: FairBatch [29] is an easy-to-use model training technique for fairness, and Slice Finder [13, 14] is an easy-to-use model evaluation technique for improving fairness.



```
fairsampler = FairBatch(model, criterion, train_data,
                        batch_size, alpha, target_fairness)
loader = DataLoader(train_data, sampler = fairsampler)

for epoch in range(epochs):
  for i, data in enumerate(loader):
    # get the inputs; data is a list of [inputs, labels]
    inputs, labels = data

    ... model forward, backward, and optimization ...
```

Figure 8: (a) The black path shows how the model fairness improves as FairBatch adjusts two parameters λ_1 (sampling rate for examples where Z=0 given Y=0) and λ_2 (sampling rate for examples where Z=0 given Y=1) for each epoch on the COMPAS dataset. "ED disparity" is the accuracy difference conditioned on the true label between sensitive groups where lower disparity means better equalized odds. (b) Sample PyTorch code where the batch selection sampler is replaced with FairBatch with a single-line change highlighted in blue.

5.1 Batch Selection for Fair Models

While many unfairness mitigation techniques [9] have been proposed, most of them require significant amounts of effort to deploy. Pre-processing techniques have the advantage of being applicable to any model, but require changes in the training data in order to remove bias. In-processing techniques tend to perform well, but usually propose a new model training algorithm that completely replaces an existing algorithm. An interesting question is whether we can take the best of both worlds of pre-processing and in-processing without their overheads.

We show that such a solution exists and propose FairBatch, which simply improves the batch selection of stochastic gradient descent training for better fairness. We formulate a bilevel optimization problem where we keep the standard training algorithm as the inner optimizer while incorporating the outer optimizer to equip the inner problem with the additional functionality: adaptively selecting minibatch sizes for the purpose of improving fairness. While the model is training, FairBatch adaptively adjusts the portions of the sensitive groups within each batch that is selected for each training epoch based on the fairness of the current intermediate model. For example, let us use the COMPAS example where we are predicting recidivism rates of criminals. Also let us use equalized odds (defined in Section 3) as the fairness measure where we want the positive prediction rates of sensitive groups to be the same conditioned on the true label. Since the label is fixed, this fairness can be interpreted as the model having the same accuracy for sensitive groups conditioned on the label. Now suppose that an intermediate model shows higher accuracy for a certain sensitive group. FairBatch then increases the batch-size ratio of the other underperforming sensitive group in the next batch. Intuitively, a larger batch size ratio results in better accuracy, so eventually equalized odds will improve. Figure 8 (a) illustrates how FairBatch improves equalized odds during a single model training. In reference [29], we show that this strategy is theoretically justified and generalize the algorithm for other fairness measures including demographic parity.

A key feature of FairBatch is its usability where one only needs to replace the batch selection of a machine learning system. Figure 8 (b) shows a PyTorch code example where one can deploy FairBatch by replacing a single line of code, and no further changes are needed in the pre-processing or in-processing steps of model training. In reference [29], we also conduct experiments on synthetic and real datasets and show that FairBatch surprisingly has performances comparable to or even better than state-of-the-art pre-processing and in-processing unfairness mitigation techniques in terms of accuracy, fairness, and runtime. In addition, FairBatch is flexible and can be used to improve the fairness of pre-trained models like ResNet18 and GoogLeNet. Finally, there are batch selection techniques proposed for faster model training convergence, and FairBatch can be naturally combined with them to improve fairness as well.

5.2 Automatic Data Slicing for Fair Models

After model training, models are evaluated before being served. For example, TensorFlow Model Analysis [4] is a model evaluation component of TFX that accepts a user-specified slicing feature (e.g., country) and shows the model accuracies per slice (e.g., accuracy per country). Here we are using equalized error rates (defined in Section 4.1) as our notion of fairness. However, there is potentially an exponential number of slices to explore, and it is not easy for users who do not have enough domain expertise to quickly sift through them.

We thus propose Slice Finder [13, 14], which automatically finds "problematic" slices (subsets of the data) where the model underperforms. Given these slices, users can take action by acquiring more data as in Slice Tuner or debug the problematic data to find the root cause that led to the poor performance. We define a problematic slice to have the following characteristics. First, the slice must be interpretable where it can be defined with feature-value pairs, e.g., "Gender=Male and Age=20-30." While one can also define a slice to be a cluster of examples, clusters are often difficult to understand in practice. In addition, the slice must have a relatively lower accuracy than its complement, i.e., the rest of the examples other than the slice, where the difference (effect size) is large and statistically significant. Finally, the slice must be large enough to have a meaningful impact on the overall model accuracy.

Since the search space for all possible slices is vast, we propose two approaches for searching. The first is a decision tree approach where we construct a decision tree of feature-value pairs to find slices. The traversal is fast, but the slices are non-overlapping, which means that we may miss some problematic slices. The second is a lattice search approach where we find slices by traversing a lattice of feature-value pairs in a breadth-first manner. Although we now find overlapping slices, this searching is slower than the decision tree approach. Once we find potential problematic slices, we perform effect-size and significance testings.

In references [13, 14], we show that Slice Finder performs better than a clustering baseline on real datasets. Also while lattice searching is slower than decision tree searching, it finds more problematic slices.

6 Open Challenges

We are far from achieving responsible AI for end-to-end machine learning and suggest promising directions. First, there needs to be deeper and broader support for the responsible AI objectives in each step of end-to-end machine learning. In addition, we believe the usability aspect of responsible AI has been largely understudied, and that there needs to be more emphasis on this important direction. Below are some concrete suggestions.

- *Data Collection*: We believe data acquisition must also support robustness. Dataset searching is becoming increasingly easy, and one challenge is distinguishing any poisoned data from the rest of the data. We also believe it is important to address fairness and robustness in data labeling.

- *Data Cleaning and Validation*: MLClean is preliminary, and an interesting direction is to develop more general and automatic cleaning and validation techniques that support various combinations of data cleaning algorithms, fairness measures, and poisoning attacks.

- *Model Training*: FR-Train is a first of its kind and can be extended in many ways. First, there needs to be more investigation on how to defend against more sophisticated poisoning attacks other than labeling flipping. Second, algorithm stability is a well-known issue in adversarial training and can be improved. Third, one may want to train models without a clean validation set.

- *Model Evaluation*: There needs to be more robustness research for model evaluation where we can easily tell whether a model is accurate enough despite data poisoning in the training data.

- *Model Management and Serving*: There needs to be more model managing and serving techniques that support fairness and robustness. While there are task-specific solutions like fairness in ranking [32], an interesting direction is to generalize and support any task with minimal configuration.
- There needs to be holistic solutions for the rest of the responsible AI objectives including explainability, transparency, and accountability. For example, recent data provenance and metadata [1, 10] solutions can be used to explain why each step in machine learning produced a certain result.

7 Conclusion

We proposed three research directions – depth, breadth, and usability – towards fully supporting responsible AI in end-to-end machine learning. While most research focuses on supporting one of many responsible AI features, we believe multiple objectives should be supported together, preferably in all steps from data collection to model serving. So far, we have scratched the surface of this vision where we proposed the following systems: FR-Train (holistic fair and robust training), Slice Tuner (selective data acquisition for fair models), MLClean (data cleaning for fair and robust models), FairBatch (easy-to-use batch selection for fair models), and Slice Finder (easy-to-use problematic slice finding for fair models). We also suggested various open challenges.

Acknowledgement

This work was supported by a Google AI Focused Research Award and by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

References

[1] Machine learning metadata. `https://www.tensorflow.org/tfx/guide/mlmd`. Accessed Jan 15th, 2021.

[2] Responsible ai practices. `https://ai.google/responsibilities/responsible-ai-practices`. Accessed Jan 15th, 2021.

[3] Responsible ai principles from microsoft. `https://www.microsoft.com/en-us/ai/responsible-ai`. Accessed Jan 15th, 2021.

[4] Tensorflow model analysis. `https://www.tensorflow.org/tfx/guide/tfma`. Accessed Jan 15th, 2021.

[5] Trusting ai. `https://www.research.ibm.com/artificial-intelligence/trusted-ai/`. Accessed Jan 15th, 2021.

[6] J. Angwin, J. Larson, S. Mattu, and L. Kirchner. Machine bias: There's software used across the country to predict future criminals. And its biased against blacks., 2016.

[7] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2019. `http://www.fairmlbook.org`.

[8] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. TFX: A tensorflow-based production-scale machine learning platform. In *KDD*, pages 1387–1395, 2017.

[9] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John T. Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. AI fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev.*, 63(4/5):4:1–4:15, 2019.

[10] Emily M. Bender and Batya Friedman. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *TACL*, 6:587–604, 2018.

[11] Omar Benjelloun, Shiyu Chen, and Natasha F. Noy. Google dataset search by the numbers. In *ISWC*, pages 667–682, 2020.

[12] Eric Breck, Martin Zinkevich, Neoklis Polyzotis, Steven Euijong Whang, and Sudip Roy. Data validation for machine learning. In *MLSys*, 2019.

[13] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Slice finder: Automated data slicing for model validation. In *ICDE*, pages 1550–1553, 2019.

[14] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Automated data slicing for model validation: A big data - AI integration approach. *IEEE TKDE*, 32(12):2284–2296, 2020.

[15] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *IEEE S&P*, pages 81–95, 2008.

[16] Tobias Domhan, Jost Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015.

[17] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *KDD*, pages 259–268, 2015.

[18] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *NeurIPS*, pages 3315–3323, 2016.

[19] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

[20] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. ACM, 2019.

[21] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.*, 33(1):1–33, 2011.

[22] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *CoRR*, abs/1811.00741, 2018.

[23] Arun Kumar, Matthias Boehm, and Jun Yang. Data management in machine learning: Challenges, techniques, and systems. In *SIGMOD*, pages 1717–1722, 2017.

[24] Andrea Paudice, Luis Muñoz-González, and Emil C. Lupu. Label sanitization against label flipping poisoning attacks. In *ECML PKDD*, pages 5–15, 2018.

[25] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *SIGMOD*, pages 1723–1726, 2017.

[26] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data lifecycle challenges in production machine learning: A survey. *SIGMOD Rec.*, 47(2):17–28, 2018.

[27] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data - AI integration perspective. *IEEE TKDE*, 2019.

[28] Yuji Roh, Kangwook Lee, Steven Euijong Whang, and Changho Suh. Fr-train: A mutual information-based approach to fair and robust training. In *ICML*, volume 119, pages 8147–8157, 2020.

[29] Yuji Roh, Kangwook Lee, Steven Euijong Whang, and Changho Suh. Fairbatch: Batch selection for model fairness. In *ICLR*, 2021.

[30] Sebastian Schelter, Yuxuan He, Jatin Khilnani, and Julia Stoyanovich. Fairprep: Promoting data to a

first-class citizen in studies on fairness-enhancing interventions. In *EDBT*, pages 395–398, 2020.

[31] Hwanjun Song, Minseok Kim, Dongmin Park, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *CoRR*, abs/2007.08199, 2020.

[32] Julia Stoyanovich, Bill Howe, and H. V. Jagadish. Responsible data management. *PVLDB*, 13(12):3474–3488, 2020.

[33] Ki Hyun Tae, Yuji Roh, Young Hun Oh, Hyunsu Kim, and Steven Euijong Whang. Data cleaning for accurate, fair, and robust models: A big data - AI integration approach. In *DEEM@SIGMOD*, 2019.

[34] Ki Hyun Tae and Steven Euijong Whang. Slice tuner: A selective data acquisition framework for accurate and fair machine learning models. In *SIGMOD*, 2021.

[35] Suresh Venkatasubramanian. Algorithmic fairness: Measures, methods and representations. In *PODS*, page 481, 2019.

[36] Steven Euijong Whang and Jae-Gil Lee. Data collection and quality challenges for deep learning. *Proc. VLDB Endow.*, 13(12):3429–3432, 2020.

[37] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[38] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *AIES*, pages 335–340, 2018.

[39] Zhifei Zhang, Yang Song, and Hairong Qi. Age progression/regression by conditional adversarial autoencoder. In *CVPR*, pages 4352–4360, 2017.